Sequence-To-Sequence Transformer Model Application

Davide Vettore 868855

1. Objective:

The objective of this project is to explore and analyze the performance of a transformer-based model on a sequence classification task. We aim to investigate how varying the sequence length and number of categories affects the model's performance metrics

2. Introduction:

Transformers have become a cornerstone in **Natural Language Processing** (NLP) due to their ability to handle sequential data efficiently. This project focuses on a transformer model applied to a sequence classification task, evaluating its performance on different sequence lengths and number of categories. The transformer architecture, known for its self-attention mechanism, allows the model to weight the importance of different words in a sequence, which is crucial for tasks like text classification.

A **Sequence-to-Sequence** task represents a task where the input and the output are sequences, not necessarily of the same length. Popular tasks in this domain include machine translation and summarization, typically involving a Transformer **encoder** for interpreting the input sequence and a **decoder** for generating the output in an autoregressive manner. In this lab, however, we simplify the task to focus solely on the encoder. Given a sequence of N numbers between 0 and M, the task is to reverse the input sequence. Although this task sounds simple, it requires handling long-term dependencies, which can be challenging for traditional Recurrent Neural Networks (RNNs). Transformers, designed to support such dependencies, are expected to perform well on this task.

We trained and tested the model on a dataset consisting of sequences with various lengths and numbers of categories, aiming to understand how these factors impact the accuracy metric. This investigation will provide insights into the transformer model's robustness and efficiency in handling sequence-to-sequence tasks.

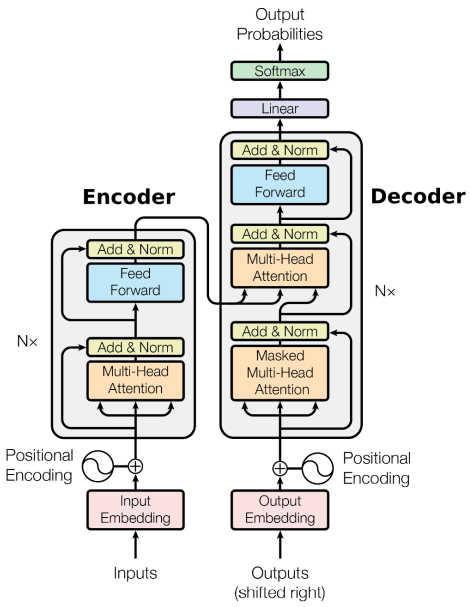

Figure 1: Typical Transformer encoder-decoder architecture

3. Procedure:

A customized dataset is defined and will be used across different data loaders for training, validation, and testing purposes. As stated before, in different runs we'll modify the sequence length and the number of categories to see how the accuracy varies.

For the **training** phase, we create a data loader that instantiates the customized dataset with 50,000 samples. The data loader is configured with a batch size of 128, shuffling the data at each epoch to ensure robust training and dropping the last incomplete batch to maintain consistent batch sizes. Additionally, we enable pin memory to speed up data transfer to GPU memory if available.

For **validation**, we set up a data loader with 1,000 samples from the customized dataset, maintaining the same batch size of 128. Unlike the training data loader, this one does not shuffle the data or drop the last batch, as these steps are unnecessary for validation.

Similarly, the **testing** data loader is configured with 10,000 samples, again using a batch size of 128. This setup ensures that our model is evaluated on a substantial amount of data to gauge its performance accurately.

The label is simply the tensor flipped over the sequence dimension. This structured approach allows us to efficiently train the model and assess its accuracy and robustness across different stages using consistent and well-prepared datasets.

An arbitrary sample of the smallest generated dataset (100 sequence length and 10 categories) is showed below:

```
Input data: tensor([9, 3, 1, 6, 8, 6, 3, 3, 2, 9, 7, 9, 2, 6, 0, 3, 5, 2, 3, 9, 3, 7,
                    5, 8,  8, 1, 3, 5, 1, 0, 4, 5, 1, 8, 0, 9, 8, 5, 1, 2, 7, 7, 4, 2,
                    2, 3, 5, 0, 5, 6, 6, 5, 9, 6, 1, 1, 7, 5, 3, 9, 0, 6, 3, 5, 7, 8,
                    8, 9, 2, 8, 5, 6, 4, 2, 6, 7, 0, 3, 6, 0, 4, 5, 4, 5, 4, 8, 0, 5,
                    2, 4, 3, 9, 4, 9, 1, 1, 0, 4, 3, 8])

Labels:     tensor([8, 3, 4, 0, 1, 1, 9, 4, 9, 3, 4, 2, 5, 0, 8, 4, 5, 4, 5, 4, 0, 6,
                    3, 0, 7, 6, 2, 4, 6, 5, 8, 2, 9, 8, 8, 7, 5, 3, 6, 0, 9, 3, 5, 7,
                    1, 1, 6, 9, 5, 6, 6, 5, 0, 5, 3, 2, 2, 4, 7, 7, 2, 1, 5, 8, 9, 0,
                    8, 1, 5, 4, 0, 1, 5, 3, 1, 8, 8, 5, 7, 3, 9, 3, 2, 5, 3, 0, 6, 2,
                    9, 7, 9, 2, 3, 3, 6, 8, 6, 1, 3, 9])
```

During training, we utilize the Transformer encoder to predict the output for each input token within the sequence. Each input number is represented as a one-hot vector, which preserves the distinctiveness of each category. This approach ensures that categories are treated as distinct entities rather than assuming any ordinal relationship between them.

The model is configured with a single encoder block and a single head in the Multi-Head Attention mechanism. This configuration is chosen due to the simplicity of the task, allowing the attention mechanism to provide interpretable explanations for the predictions.

The training is performed using the **Cross-Entropy loss function**, which is well-suited for classification tasks involving multiple classes, over a maximum of 20 epochs.

4. Results:

In Table 1, the validation and test accuracies obtained by training the transformer on different custom datasets are shown. As expected, we can see that both increasing the sequence length and the number of categories result in lower accuracy.

Sequence length	Number of categories	Validation accuracy	Test accuracy
100	10	100%	100%
100	20	100%	100%
200	10	85.25%	85.27%
200	20	48.68%	48.70%
500	10	10.05%	10.03%
500	20	4.96%	5.01%
1000	10	9.96%	10.00%
1000	20	5,00%	5,01%

Table 1: Performance of the model on multiple custom datasets

It is also possible to display the attention maps of our trained transformers for the reverse task. The plotting function takes as input the sequences, attention maps, and an index indicating for which batch element we want to visualize the attention map; over rows, we have different layers, while over columns, we show the different heads.

In Figure 2, the attention maps of the transformer model trained on three different custom datasets with various accuracies are reported. Starting from the left, we have the attention map obtained by the model trained on the dataset with sequences of 100 numbers and 10 categories, then the one obtained on the dataset with 200 sequence length and 20 categories, and finally the one obtained on the dataset with 1000 sequence length and 20 categories.

As we can see, with short sequences, the model has learned to attend to the token that is on the flipped index of itself, as intended. The longer the sequence, the harder it gets to understand the flipped relationship. The model trained with a sequence length of 200 and 20 categories (which achieves almost 50% test accuracy) seems to pay attention to certain values it's not supposed to, while the model trained on the bigger dataset doesn't learn anything.

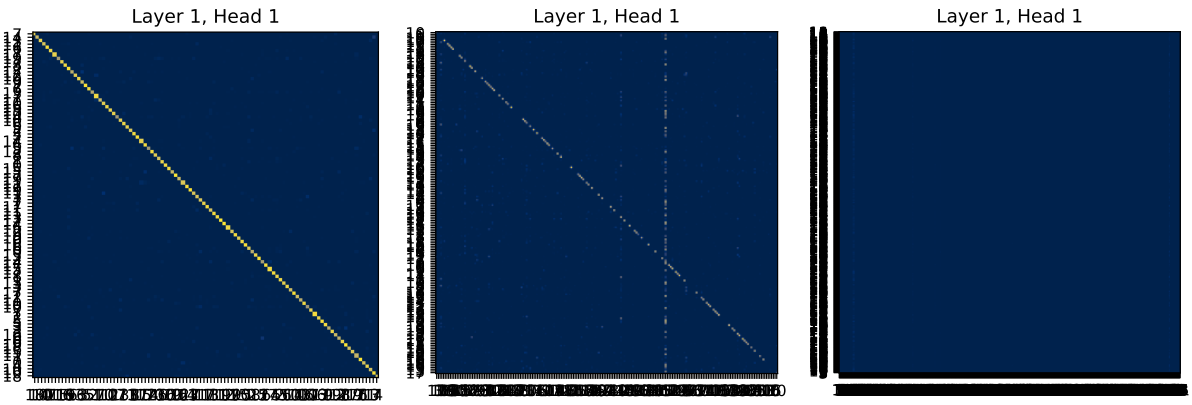

Figure 2: Attention maps of the transformer trained on three different custom datasets

5. Conclusions:

In this project, we explored the performance of a transformer-based model on a sequence classification task, specifically focusing on how varying sequence lengths and the number of categories affect accuracy. Our findings show that transformers are effective for handling sequential data, thanks to their self-attention mechanism, which is crucial for tasks involving long-term dependencies.

The results demonstrated that the transformer model performs well on shorter sequences with fewer categories, achieving 100% accuracy for sequences of length 100 with 10 and 20 categories. However, as the sequence length and number of categories increased, the model's accuracy decreased significantly.

The attention maps provided insights into how the model's attention mechanism functions across different datasets. For shorter sequences, the model correctly identified the flipped relationship between input and output tokens. However, for longer sequences, the model struggled to learn this relationship, as seen from the less focused attention maps.

In summary, while transformers are powerful for sequence-to-sequence tasks, their performance is heavily influenced by the length of the sequences and the number of categories involved. The model excels with shorter sequences but faces challenges with longer ones, highlighting the need for further improvements or alternative approaches for handling more complex sequence tasks, such as employing more sophisticated transformer architectures.