SAPIENZA
UNIVERSITÀ DI ROMA

Sapienza, Università di Roma

Master in Artificial Intelligence and Robotics

Deep Learning Project

Author:
Matteo Zaramella 2025806
Lapo Carrieri 2027737

13th October 2023

Table of Contents

1 Introduction

This report provides the details of the training and evaluation done with different models for the Deep Learning Project. Our project consists of training a State of the art, baseline and a personalized model on the Math Dataset for Question Answering. This task could seem very easy for a human being, but it is way more difficult for an artificial intelligence model. In fact, the models are trained to understand in some ways the question given as input and predict the next character based on it.

2 Encode the Dataset

To train the models, we needed to encode the dataset, converting each word in tokens. To do so, we utilized at the start the Bert Tokenizer. After some experiments, we decided to move to a custom encoding based on characters, because the Bert Tokenizer, been trained for way more complex texts, has a lot of tokens and a huge vocabulary. So we implemented our encoding based on characters. Each character is associated with an id (numeric id) and the vocabulary is made by all the characters present on the Training, Validation, and Test set. We added also some special characters like, "¡start¿" and "¡end¿" to well define each question and answer, "PAD" to add padding, and "UNK" for the unknown characters.

3 Models

3.1 LSTM (baseline)

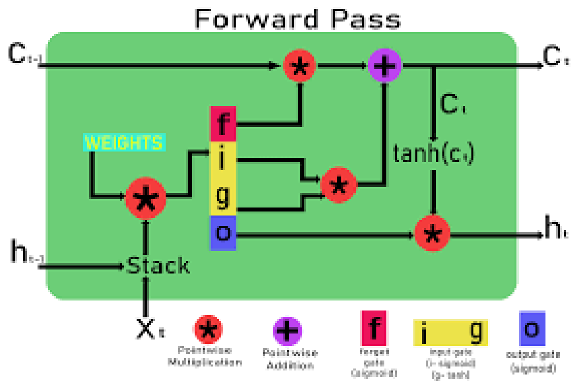


Figure 1: LSTM scheme

The first model we built is the Long Short Term Memory (LSTM) Figure 1 network model as a baseline. LSTM was chosen due to its capability of capturing sequential dependencies and due to its popularity in natural language processing tasks. We tried different architectures for this model: at the start converting the question (each token) in one hot encoder vector and trying to predict one character per time. This approach didn't reach very good results, nor did it change the tokenization method, maybe due to the training limitations of using Colab. After many trials, we decided to adopt a very simple and easy model based on only an LSTM layer. This model doesn't reach great results as well but it is still better than the ones before, and we thought was good as a baseline.

3.2 TP-Transformer (state of the art)

The TP-Transformer Figure 5 is known for its excellent performance in understanding questions with mathematical context. The accuracy and precision reached from this model in the math Dataset are the best, for this reason, it is considered the State of the art in this task. Unlike normal LSTM models that have been previously used, the TP-Transformer uses the addition of the self-attention mechanism. The `SelfAttention` class at `model/tq_soft.py` contains the implementation of the self-attention mechanism. It allows the model to weigh the importance of different parts of an input sequence when making predictions or generating output. This mechanism is the backbone of the TP-Transformer; it facilitates the model to understand the relationship between the input elements in a much more effective and efficient manner. Self-attention is the main feature that allows the model to answer a variety of tasks including mathematics reasoning as well as answering questions of all orders.

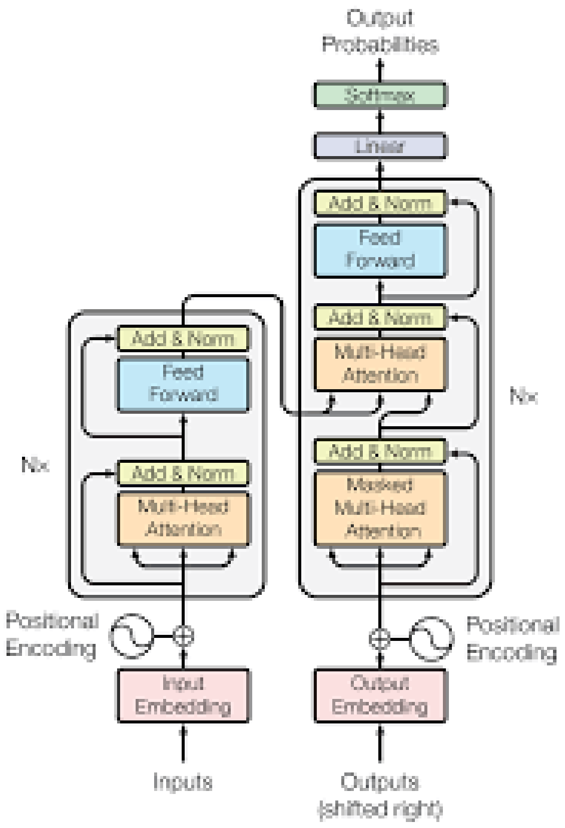


Figure 2: Transformer scheme

3.3 Transformer with Hybrid Attention (our model)

We propose an alternative to the TP-TRansformer, changing the Self-attention mechanism present on it with the Hybrid Attention mechanism. It consists of the combination of multihead attention and additive attention, to extract the most relevant information and apply it to the mathematical questions.

Multihead attention allows a model to focus on different parts of the input sequence in parallel, enabling it to capture diverse relationships and patterns within the data. additive attention is an attention mechanism that transforms query and key vectors before computing compatibility scores, this provides the model with additional flexibility in learning complex relationships between elements in a sequence.

More specifically: Multihead attention consists of applying the attention layer (made by Key Query and Value) several times in parallel. For our model, we used 8 heads, so it could be focused on more parts of the input in parallel.

Additive attention, instead, is calculated by doing a hyperbolic tangent and softmax function.

1. Linear Transformations: The query (Q) and key (K) vectors are linearly transformed using learned weight matrices (W_Q and W_K):

   $$Q' = Q \cdot W_Q \quad \text{and} \quad K' = K \cdot W_K$$

   where Q' and K' are the transformed query and key vectors respectively.

2. Adaptive Score: The transformed query and key vectors are then combined to produce a compatibility score:

   $$\text{Adaptive Score} = \tanh(Q' + K') \cdot W_V$$

   where W_V is another learned weight matrix.

3. Additive Attention Formula: The adaptive scores are then used to calculate the attention. d_k is the dimension of the key vectors, it is used to scale down the dot product to prevent it from becoming too large:

   $$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

4 Evaluation and conclusion

We did several experiments but without reaching very satisfy results. We trained each model 30 epochs with training set made by 100000 samples and a test set of 10000, and we obtained the results shown on Table 1.

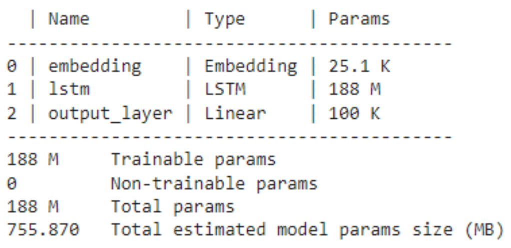
In general is evident the problem in our model, the accuracy is always low since the dataset is too small or for problems related to the code. We wanted to train a better

Model	Epochs	Test Accuracy	Final Loss
LSTM	30	0.4172	0.04
TP	30	0.2357	1.7201
Our Transformer	30	0.1743	1.923

Table 1: Training Results

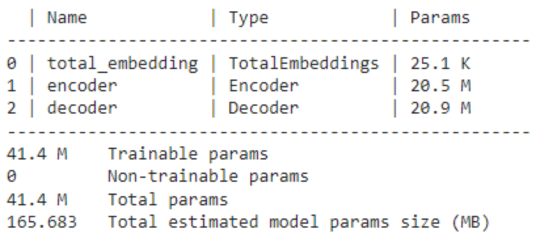
model with a bigger training set but we weren't able due to the colab limitation. For the same reason our results are not good, but we think that with more samples and time the models would reach way better results. To demonstrate this, also the TP-TRansformer, that is the state of the art in this task, reaches still very low results. [In the folder shared there is the final project called "DeepLearningProject" and other draft of the works.]

4.1 Model overview in Pytorch lightning



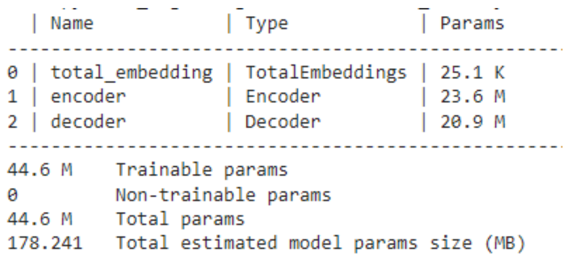
```
  | Name         | Type      | Params
-------------------------------------------
0 | embedding    | Embedding | 25.1 K
1 | lstm         | LSTM      | 188 M
2 | output_layer | Linear    | 100 K
-------------------------------------------
188 M     Trainable params
0         Non-trainable params
188 M     Total params
755.870   Total estimated model params size (MB)
```

Figure 3: LSTM scheme



```
  | Name            | Type            | Params
------------------------------------------------------
0 | total_embedding | TotalEmbeddings | 25.1 K
1 | encoder         | Encoder         | 20.5 M
2 | decoder         | Decoder         | 20.9 M
------------------------------------------------------
41.4 M    Trainable params
0         Non-trainable params
41.4 M    Total params
165.683   Total estimated model params size (MB)
```

Figure 4: TP transformer scheme



```
  | Name            | Type            | Params
------------------------------------------------------
0 | total_embedding | TotalEmbeddings | 25.1 K
1 | encoder         | Encoder         | 23.6 M
2 | decoder         | Decoder         | 20.9 M
------------------------------------------------------
44.6 M    Trainable params
0         Non-trainable params
44.6 M    Total params
178.241   Total estimated model params size (MB)
```

Figure 5: Our Transformer scheme