Final Project Step 3 - Technical Report

Jhonata Miranda da Costa
IT Department – Viçosa – MG
Federal University of Viçosa (UFV)
jhonata.miranda@ufv.br

1 Introduction

Processing tabular data is central to many machine learning applications, with ensemble-based decision tree methods, such as gradient boosted decision trees (GBDTs)(1) being widely regarded as the most effective today. However, there is growing interest in deep learning techniques, which eliminate the need for manual feature engineering. Neural network models such as DNF-Net, TabNet, and MLP+ have demonstrated performance comparable to GBDTs. Attention-based architectures, such as TabTransformer(2), leverage Multi-Head Self Attention blocks to model relationships between categorical features, significantly enhancing performance compared to pure MLP networks. The GatedTabTransformer(3), introduced in a recent paper [1], improves upon the TabTransformer by incorporating a gated multi-layer perceptron (gMLP)(4) block and explores architectural design decisions through hyperparameter optimization experiments.

To evaluate the architectures, three datasets were used: *Blastchar* [2], *1995_income* [3], and *bank_marketing* [4]. These datasets, which contain between 14 and 19 features and range from 7,000 to 45,000 samples, are designed for binary classification tasks and were also included in the original TabTransformer study. Data was split into training (65%), validation (15%), and testing (20%) sets. The results show that the GatedTabTransformer outperforms both the original TabTransformer and traditional MLP networks, highlighting the benefits of the architectural modifications.

The experimental results indicate that the GatedTabTransformer achieves improvements of 0.5% to 1.1% in mean area under the receiver operating characteristic curve (AUROC) compared to the baseline TabTransformer and 1% to 2% compared to traditional MLPs across the three datasets. The proposed model not only enhances classification accuracy but also provides a more efficient approach to modeling tabular data. This demonstrates its potential to bridge the gap between conventional machine learning methods and modern deep learning techniques, making it a promising alternative for such applications.

2 Implementation Details

2.1 Code Structure

The implemented code was divided into 3 main parts, in addition to the training implementation in the main function. This makes it easier to understand which part of the code belongs to each model used in the GatedTabTransformer architecture. This model contains 2 main points in addition to the input and output processing. The first is the use of N encoder layers to process the embeddings extracted from the categorical features. The second is the gMLP for processing the encoder output concatenated with the continuous values after a layer normalization process.

[1] https://github.com/radi-cho/GatedTabTransformer

[2] https://github.com/radi-cho/GatedTabTransformer/blob/master/data/blastchar.csv

[3] https://github.com/radi-cho/GatedTabTransformer/blob/master/data/income_evaluation.csv

[4] https://github.com/radi-cho/GatedTabTransformer/blob/master/data/bank-full.csv

The Transformer was divided into 5 other classes to better modularize the code, similar to what was done in the original paper. Following the order of the code, first we have Skip Connection, responsible for making the residual connection between the output of the attention layer and applied again after the output of the feed forward, both after passing through the next class. The second is normalization, where we have the application of layer normalization as proposed in the original article. Next, we have the implementation of the gated GELU, which replaces the RELU function of the original transformer. This activation function was introduced into this architecture by the GatedTabTransformer paper to give more flexibility to the activation mechanism. Next up is the feed forward, which is also present in the original transformer encoder architecture. Finally, we have the implementation of multi-head self-attention, in the same way as in the original encoder.

The gMLP was divided into 4 parts, the first two being residual connection and layer normalization, as in the encoder. The Gating Unit modulates the input by dividing it into two parts: residual (first half) and gate input (second half), which is normalized and processed by a learned weight matrix. If configured with causal masking, the unit ensures that positions in the sequence do not depend on future values, while the circulant option uses cyclical shifts to optimize the weight matrix. The gate input is multiplied by the weights (or by the circulant version), added to a bias, and reorganized for multiple heads in the style of multi-head attention. The final output is activated by a specified function and multiplied element by element by the residual part, preserving information from the original input while applying a learned transformation. The unit is efficient for controlled modulation on sequential or tabular data, combining the flexibility of gating mechanisms and residual connections. Next, we have the gMLP block, which modularizes a gMLP block that will be repeated a few times depending on the depth parameter of the gMLP.

2.2 Pytorch

The Pytorch framework was essential for reproducing the paper. Thanks to it, it was possible to concentrate on the reconstruction of larger concepts, such as: Multi-Head Self Attention, the use of an encoder as a layer of the architecture, Spatial Gating Unit, Gated GELU. To do this, we used some classes with the implementation of some other concepts, such as: Linear, GELU, Dropout and also functions such as matmul, permute, view, softmax and sigmoid. Other frameworks, such as Numpy, Scikit-Learn and Pandas were used, but to perform more specific calculations, such as obtaining the AUC score.

2.3 Differences to original paper

The original work provides a more complex implementation to understand, such as using the Einops library. This makes some matrix multiplications more elegant, but less readable for programmers unfamiliar with the library. They also use a specific function of this library, Rearrange, to resize a tensor. Furthermore, in addition to the GatedTabTransformer model, there is the implementation of the TabTransformer and hyperparameter optimization. The implementation that accompanies this work uses the classic method for matrix multiplication, as well as providing an alternative to the Rearrange function. In gMLP, where this function is used, Unflatten, from the Pytorch library, was applied to perform the same tensor resizing procedure. Finally, this paper only presents the implementation of the GatedTabTransformer model, without comparisons or optimizations.The original implementation inspired the organization of the code, the training method, and data pre-processing, where much of what the authors of the original work did was reused.

3 Experimental Setup

The setup used to run the experiment completely followed all the hyperparameters used in the original paper in order to obtain close results. A dictionary called config was created to store the hyperparameters in a single place to make it easier to change them when necessary. As the original paper optimized the hyperparameters for the datasets, these obtained good results when the experiment was run. The hardware used was an Nvidia GTX 1050 GPU with 3GB of memory, an Intel i5-9300HF CPU, and 16GB of RAM. Compared to the TabTransformer experiment, which used an NVIDIA V100 GPU, 8 CPUs, and 60 GB of memory, there is a significant difference in computing power. However, due to the size of the datasets, the experiment was able to be carried out

with the same hyperparameters as the original paper. The setup used by the original paper has not been disclosed.

For training, we used a batch size of 256 and an initial learning rate of 0.001, which will decrease in a scheduler with a gamma value of 0.1 and a step value of 8. The optimizer used was AdamW with amsgrad, a variation of this algorithm, and the loss chosen for the experiment was Binary Cross-Entropy due to the binary classification in all datasets. As hyperparameters for the models, we have the number of encoder heads equal to 8 and the number of layers equal to 6, as well as the value 8 for the encoder dimension. For gMLP, we defined 6 layers of depth and a dimension of 64. Talking about regularization, a dropout value was set for the attention layers, equal to 0.2. A maximum of 200 epochs was used for training, using Early Stopping with 5 epochs of patience and a log value of 10 steps.

4 Results

After carrying out the experiment, the following results were obtained, as shown in the table 1. The metric used for evaluation is the AUC score, the area under the ROC curve. This metric was used because it generalizes very well to other metrics such as precision, recall, and F1-Score, as well as being the metric used in the original work. The table shows that the results achieved by the implementation were close to the original article, with a difference in the Bank Marketing dataset where the original article achieved much better results than the present work. The table also shows the results obtained by running the TabTransformer and GatedTabTransformer models from the original experiment, generated by running the original code related to the paper.

Table 1: Model Performance Comparison

Model	**Blastchar**	**Income**	**Bank**
My implementation	0.8288	**0.8968**	0.6926
Gated TabTransformer	**0.8363**	0.8916	**0.7456**
TabTransformer	0.8320	0.8785	0.5326

5 Conclusion

Some of the notable challenges that arose while reproducing the GatedTabTransformer include achieving results close to the original implementation, especially on the Bank Marketing dataset. Even while keeping the same hyperparameters and maintaining consistency in the experimental setup, discrepancies in the results have shown the sensitivity of deep learning models to minor differences in implementation and resource constraints. This reflects the significance of computational power and library-level optimizations in reproducing deep learning models with state-of-the-art performance.

One of the other challenges was to find an alternative for the Einops `Rearrange` function, which is used extensively in the original work. Though the library simplifies tensor operations, its abstraction can result in dimensional inconsistencies if not carefully implemented. In solving this, the `Unflatten` from PyTorch was taken over. It provided an alternative much more transparent, though very functional, to resize any tensor. The deepened understanding of the gMLP block, how that acts in conjunction with other blocks of the architecture, has been fundamental. The main novelty introduced is the Gating, which complicates things rather in the way of modulating the inputs and retaining sequential information. These challenges, when overcome, provided experience in handling both the intricacies of transformer-based architectures and the practical limitations of model reproduction.

References

[1] Zhiyuan He, Danchen Lin, Thomas Lau, and Mike Wu. *Gradient Boosting Machine: A Survey*. CoRR, abs/1908.06951, 2019. Available at: `https://arxiv.org/abs/1908.06951`.

[2] Xin Huang, Ashish Khetan, Milan Cvitkovic, and Zohar Karnin. *TabTransformer: Tabular data modeling using contextual embeddings*. CoRR, abs/1908.06951, 2019. Available at: `https://arxiv.org/abs/1908.06951`.

[3] Radostin Cholakov and Todor Kolev. *The GatedTabTransformer. An enhanced deep learning architecture for tabular modeling*. CoRR, abs/2201.00199, 2022. Available at: `https://arxiv.org/abs/2201.00199`.

[4] Hanxiao Liu, Zihang Dai, David R. So, and Quoc V. Le. *Pay Attention to MLPs*. CoRR, abs/2105.08050, 2021. Available at: `https://arxiv.org/abs/2105.08050`.