Not Transformer Model(NT Model)

Beyond Transformer based GPT Model

How to build the best small LLM

Best low cost LLM build method

memonode

Not Transformer, Not Attention Model

NT-model is a time-series neural network model that does not use attention operations.

Attention operations exponentially increase the amount of computation as the sequence and dimension increase, which is the main cause of high-cost learning.

Until now, there has been no model that surpasses self-attention concentration in long-term dependency, but this model does not use attention operations and shows superior learning convergence speed and execution speed compared to Transformer GPT, and approximately 200% advantage in low memory usage, which causes an exponential difference as the amount of tokens and sequence length increases.

The computational cost increases linearly with sequence length and dimension expansion, making it easy to expand compared to Transformer, and the cost is greatly reduced, so there are no restrictions on ultra-large-scale learning and inference.

It shows excellent inference ability compared to attention in non-verbal numeric time-series data.

LLM (Language Pre-Learning GPT Comparison)

- As a result of pre-training the KorQuAD(1.0) corpus by chunking it into 512 token sequence units, the GPT model shows a loss error of 0.6596 at 800 forks and the recall of the training data is 85%.

- In the case of the NT-model, the loss error reaches 0.3819 at 200 epochs, showing a fast learning convergence speed and the recall of the training data is 99%.

- As a result of pre-training the 2023 newspaper article corpus (2.16G) as a 4K token sequence, GPT exceeds 80G of H100 memory even with a 1-layer configuration, making training impossible with 1 H100. When training with a length of 512 by fully utilizing the constraints of one H100, the training error converges to 3.80. NT-model requires 66G of activation memory when learning newspaper article data with 4K token sequences and 20 layers, so it can pre-learn 2.3 billion parameters with H100. When learning with a length of 512 under the constraint of one H100, it converges with a learning error of 3.74.

- In the above example, regardless of the vocabulary size, NT-model has better learning convergence than the transformer model, and under the constraint of one accelerator, the learning execution speed is about twice as fast and uses ½ less activation memory, and this difference increases exponentially as the number of tokens increases.

- Since the memory of the NT-model increases linearly, not quadratically, as the token sequence length increases, LLM learning is possible with one multi-GPU server.

- While SLMs such as Pi3Mini, which supports 3.8 billion parameters and 4k token sequences, require 100 H100s for pre-training, our foundation model can be trained with just a few accelerators, and the difference increases exponentially in proportion to the increase in training parameters and token sequence length, fundamentally solving the cost problem associated with current large-scale LLM pre-training.

Sign Curve (Comparing the accuracy of predicting nonverbal numeric data)

- Half of the total length of the test data is given as input and the remaining half is predicted by auto regression.

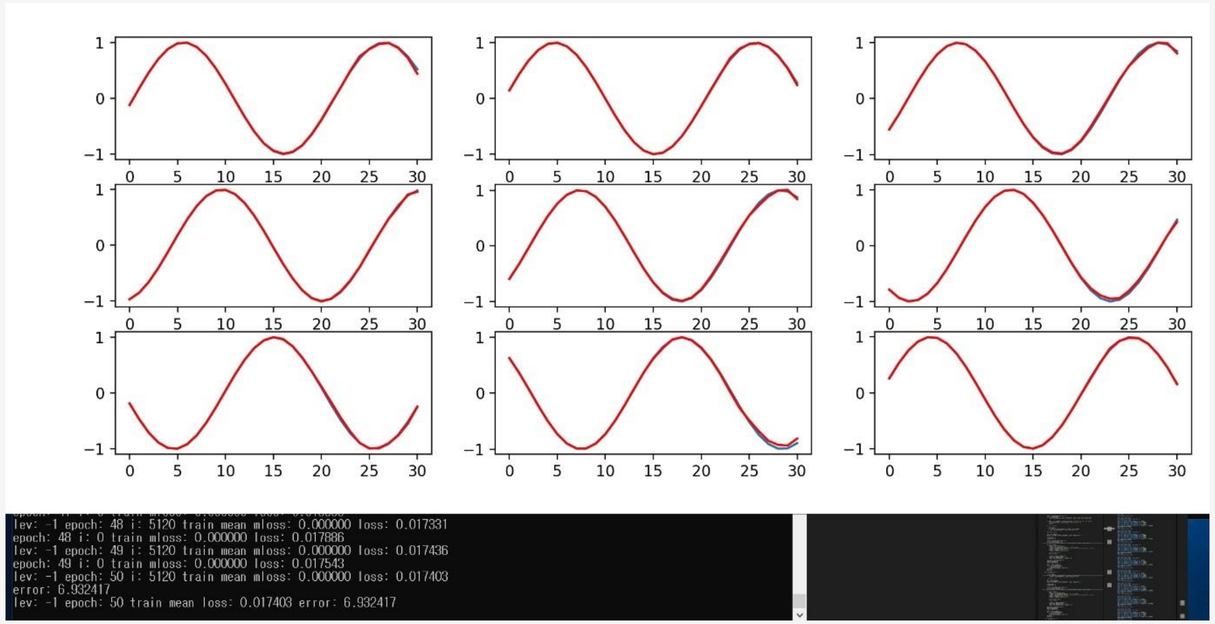
- Compared to GPT, both visual identification of the graph and mean square error in the test data set inference are accurate in NT-model.

- In cases where the regularity is clear like this, accurate learning and inference should be possible, but when looking at the learning error, the error of the transformer series model does not continue to converge and oscillates, reducing accuracy, and the test set experimental results graph also shows a slight error with the naked eye. In contrast, the NT-model model shows a precise inference result while the error continues to converge.

- Red is the predicted value, blue is the correct answer. In the case of GPT, blue appears slightly, but in the case of NT-model, it completely overlaps and only red is visible.

Sign Curve (Comparison graph of prediction accuracy of non-verbal numeric data)

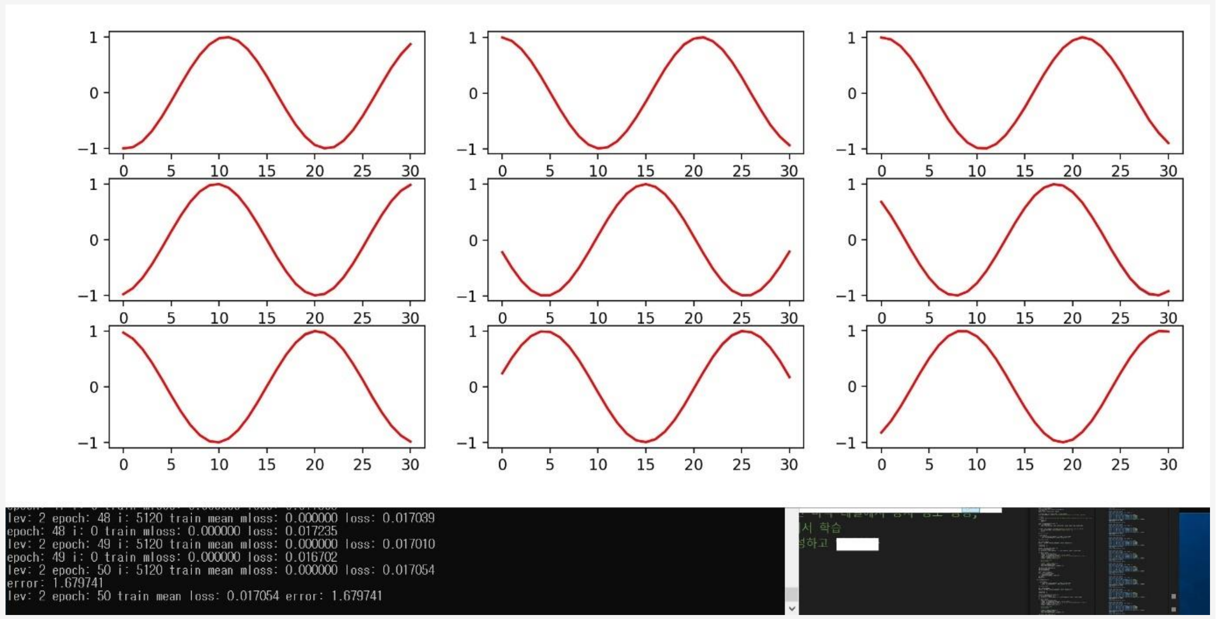
GPT



```
lev: -1 epoch: 48 i: 5120 train mean mloss: 0.000000 loss: 0.017331
epoch: 48 i: 0 train mloss: 0.000000 loss: 0.017886
lev: -1 epoch: 49 i: 5120 train mean mloss: 0.000000 loss: 0.017436
epoch: 49 i: 0 train mloss: 0.000000 loss: 0.017543
lev: -1 epoch: 50 i: 5120 train mean mloss: 0.000000 loss: 0.017403
error: 6.932417
lev: -1 epoch: 50 train mean loss: 0.017403 error: 6.932417
```

Error: 6.94471

NT-model



```
lev: 2 epoch: 48 i: 5120 train mean mloss: 0.000000 loss: 0.017039
epoch: 48 i: 0 train mloss: 0.000000 loss: 0.017235
lev: 2 epoch: 49 i: 5120 train mean mloss: 0.000000 loss: 0.017010
epoch: 49 i: 0 train mloss: 0.000000 loss: 0.016702
lev: 2 epoch: 50 i: 5120 train mean mloss: 0.000000 loss: 0.017054
error: 1.679741
lev: 2 epoch: 50 train mean loss: 0.017054 error: 1.679741
```

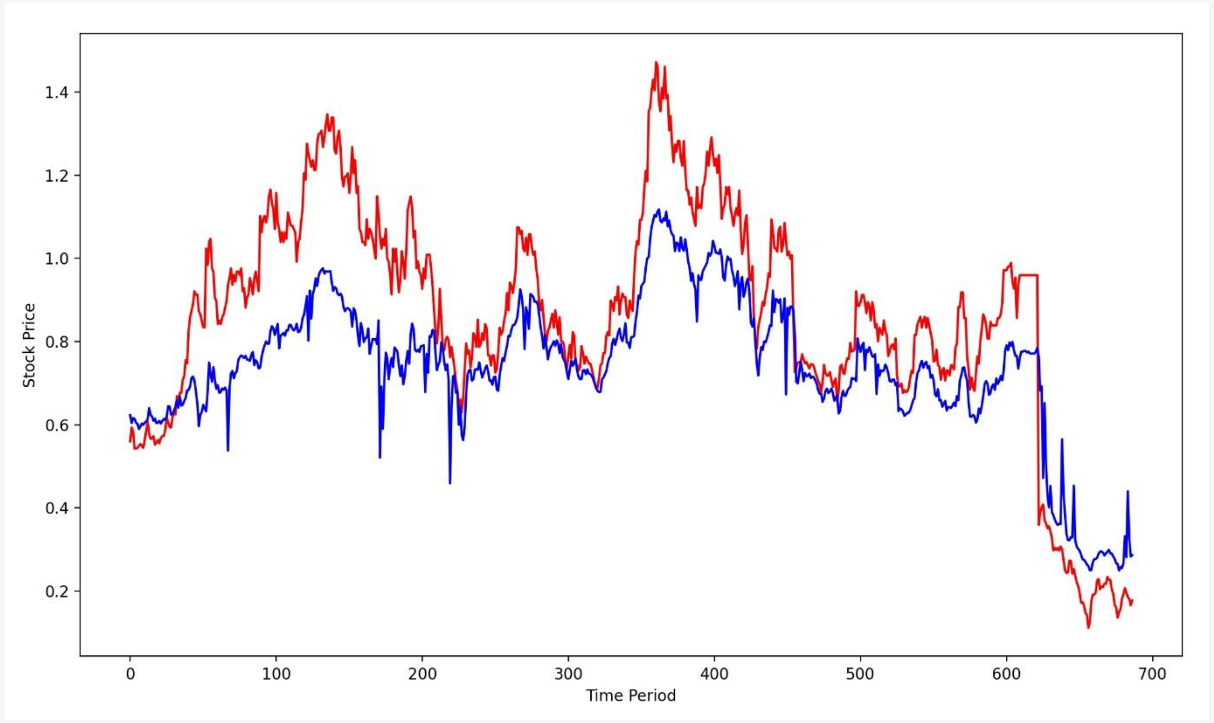
Error: 1.679741

Stock Price Prediction (Comparing Nonverbal Numerical Data Inference Ability)

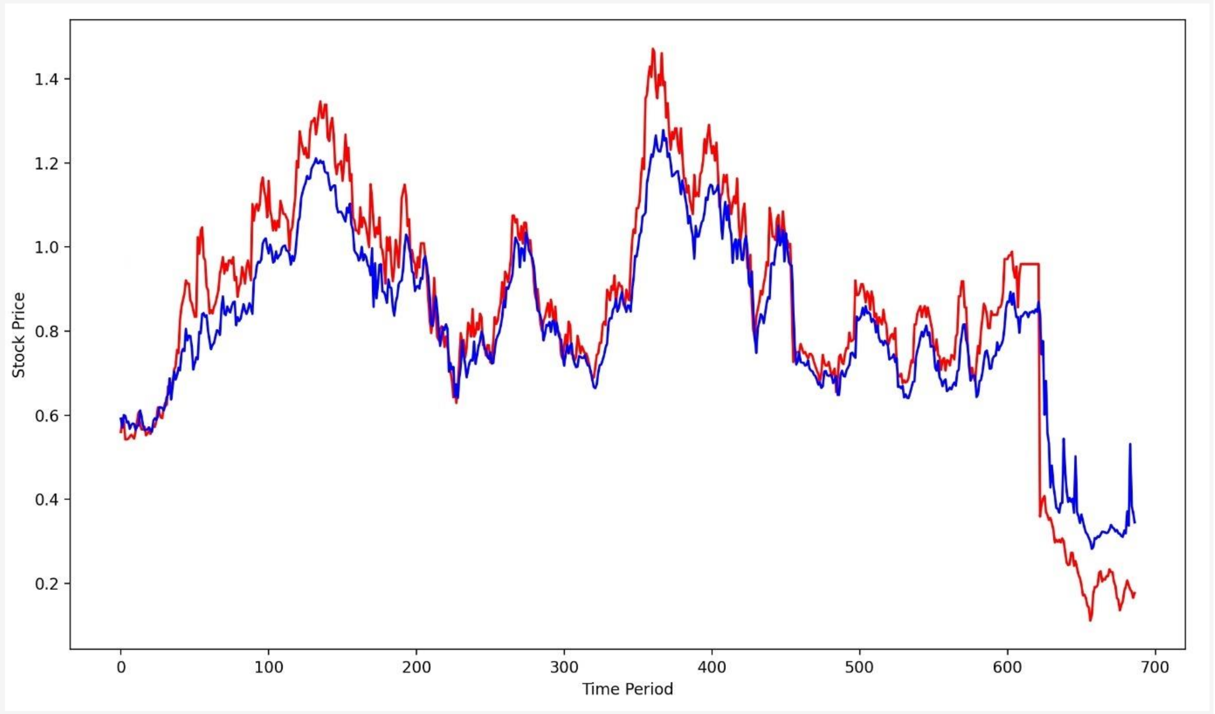
- In the case of learning and predicting with past trend data, unlike the general case, the input data does not include information about the target stock price and future values, so it is difficult to predict because there is no lagging input value to follow, so it learns and predicts the closing price of the next day. It shows an inference result that is closer to the target value than the case of GPT.

Stock Price Prediction (Comparison Graph of Nonverbal Numerical Data Inference Ability)

GPT

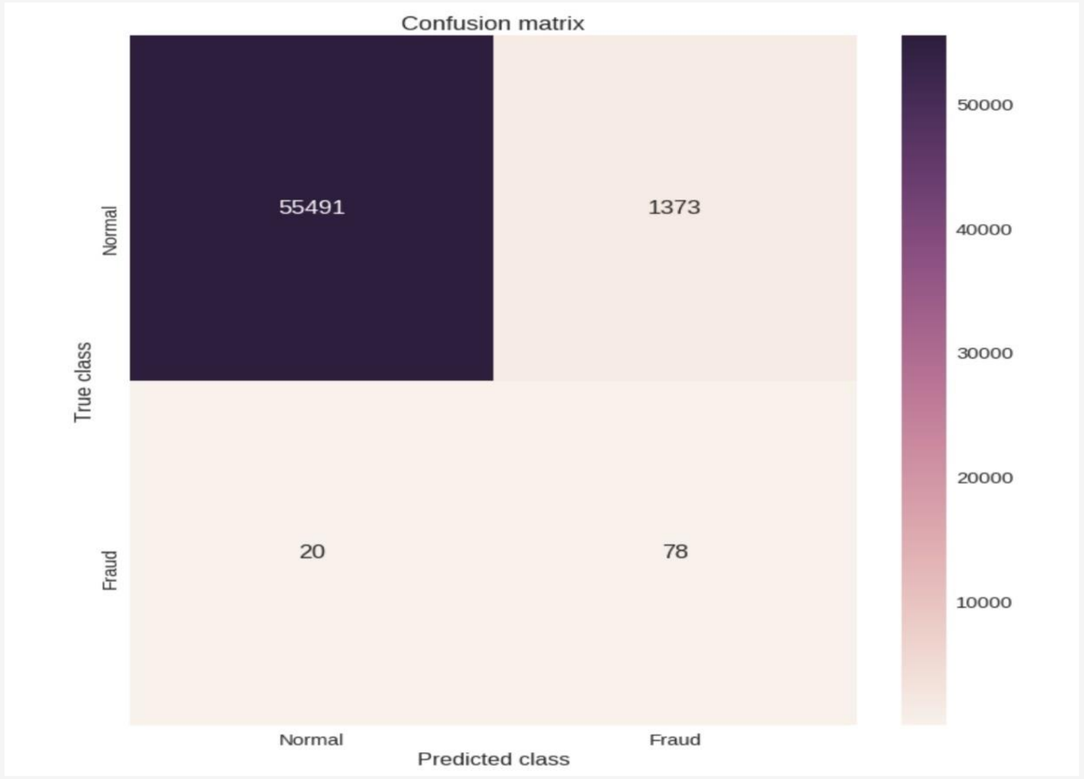


NT-model

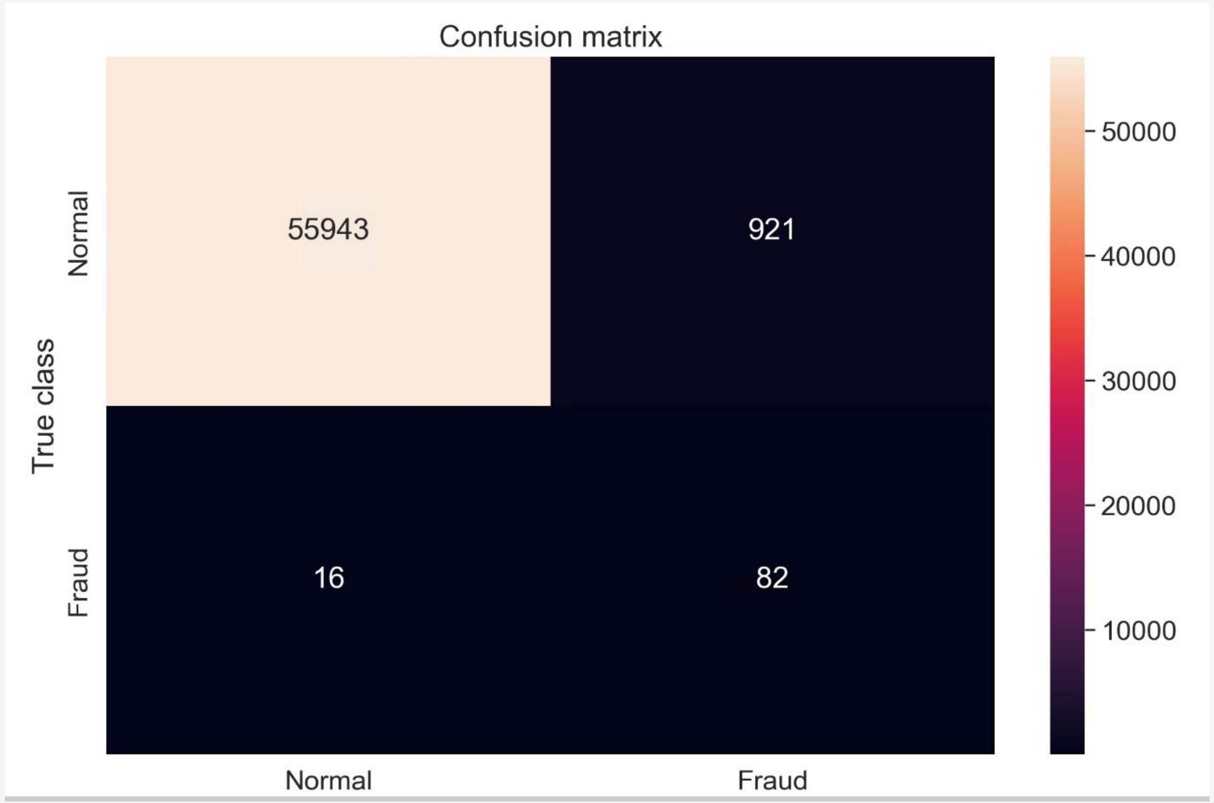


Anomaly Detection (Anomaly detection, rare data learning inference ability)

Auto Encoder



NT-model



The larger the values in the diagonal direction from the upper left to the lower right, and the smaller the values in the diagonal direction from the upper right to the lower left, the better the detection ability.

Vertical: Correct class

Horizontal: Predicted class

thank you

All examples are on GitHub: mempute (github.com)

Ai framework: mempute/flux: AI framework, c++ & python 1: 1 matching, API supporting host and gpu(cuda) libraries, powerfull time series neural net (github.com)

LLM: mempute/v4: Time series models built with PyTorch C++ and supporting Python interfaces, including: Spiking model using som(self-organizing map), quantum attention model, Bayesian inference network using rdbms (github.com)

LLM pre-training results: mempute/learn_results (github.com)

If you want an alternative to transformer-based LLM or want to differentiate inference performance from traditional attention on non-language time series data, please contact me.

For other inquiries, please contact mempute@gmail.com.