Sebastian Teslic, Jason

12/8/2025

Math 495 Machine Learning

Prof. Khaitan

Learning Numerical Sequences with Transformer Models

Abstract

This project investigates the capability of a hybrid transformer architecture for learning and predicting numerical sequence patterns. We generated a dataset that labeled ten distinct sequence types: constant, linear, quadratic, logarithmic, prime numbers, Collatz sequences, arithmetic progressions, Fibonacci sequences, noisy sinusoidal patterns, and geometric progressions. Our model extends a standard transformer encoder by incorporating multiple prediction heads (delta, ratio, and direct value), type embeddings, and a learned gating mechanism that combines prediction strategies. Training is performed by normalizing the synthetic dataset to have zero mean and standard deviation one. Then through the sliding window technique we ask our network to make a prediction, then compute the errors and adjust the weights to minimize these errors. Experimental results show that the transformer architecture accurately captures patterns in deterministic sequences and demonstrates partial success on chaotic sequences (Collatz, Fibonacci, prime numbers), revealing limitations due to the irregular structure. This study demonstrates that transformers can serve as sequence learners, however these come with limitations.

Introduction

Numerical sequences are fundamental objects in mathematics that appear naturally in many areas of science and engineering. They range from deterministic, predictable patterns such as arithmetic progressions to complex structures like prime numbers. This project explores whether a single, unified machine learning model can learn to recognize and predict the behavior of multiple sequence types simultaneously without being explicitly programmed with their rules. Transformers, originally developed for natural language processing, use self-attention mechanisms to capture long-range dependencies in sequential data.

To test this capability and versatility, we selected ten diverse types of sequences: deterministic sequences with closed-form solutions (constant, linear, quadratic, arithmetic, geometric), recursively-defined sequences (Collatz, Fibonacci), along with primes, logarithmic sequences, and noisy sinusoidal sequences. These sequences span a wide range of behaviors, from smooth, predictable growth to chaotic behavior. The diversity of this dataset provides a rigorous test for the transformer to function as a general-purpose sequence learner.

The scope of this project is to train a lightweight transformer-based model to accurately predict subsequent values in each sequence and to analyze which classes of sequence patterns the transformer learns most successfully.

Background

The transformer architecture fundamentally transformed the task of sequence modeling. Transformers have powered models like BERT and ChatGPT, and also applications beyond text such as ViT (Vision Transformer). Transformers use self-attention mechanisms that allow parallel processing. These self-attention mechanisms allow the model to weigh relationships

between all positions in a sequence. With multi-head attention, multiple attention mechanisms run in parallel, with each learning different types of relationships within the data. Positional encodings are added to input embeddings to include information about the order of elements. Feedforward networks function after attention layers to add non-linear transformations. Layer normalization and residual connections stabilize training and allow smooth gradient flow.

Numerical sequences pose a set of unique challenges: unlike natural language processing, mathematical sequences are often shorter, providing fewer examples for the model to learn from. Large growth rates require normalization strategies that manage scale and preserve patterns. Autoregressive patterns are prone to accumulating errors, especially for such sequences. Smooth algebraic sequences are easy to fit because patterns are consistent across the entire sequence; in contrast, prime numbers lack this same simple structure, exhibiting discontinuous jumps. Fibonacci numbers require learning a long-range recurrence pattern, and Collatz numbers are highly sensitive to initial conditions. These properties make numerical sequences ideal for testing whether transformer architectures can learn mathematical structure implicitly.

Dataset Generation and Preprocessing

We generated a synthetic dataset containing ten distinct mathematical sequence types. Each sequence type was generated using its mathematical definition. The sequence types include:

1. Constant sequences: All elements have the same value.
2. Linear sequences: Arithmetic progressions defined by $a_n = a_0 + dn$.
3. Quadratic sequences: Defined by polynomial growth.
4. Logarithmic sequences: Values increase according to $a_n = log(n + c)$.

5. Prime sequences: Consisting of consecutive prime numbers.
6. Collatz sequences: Generated using the Collatz iteration rule.
7. Arithmetic sequences: Sequences with fixed additive increments.
8. Fibonacci sequences: Defined recursively.
9. Noisy sinusoidal sequences: Obtained by adding noise to a sinusoidal curve.
10. Geometric sequences: Sequences with multiplicative growth.

For each sequence type, the sequences were generated with varying parameters (e.g., slope, coefficients, amplitude) to ensure diversity. This dataset was stored in a text file format in which each line contained the sequence type label followed by the type ID, followed by comma-separated numerical values. Since the dataset contains sequences that vary in scale and growth rate, we used a normalization strategy to ensure stable training and avoid the loss function being dominated by sequences of large magnitude. For each raw sequence, we computed its mean and standard deviation, and then normalized each element.

In order to train the model to predict the next value in a sequence, we used a sliding window generation technique. For each window of length $L = 30$, each sequence was segmented into overlapping samples by shifting the window one position at a time. This approach generated multiple training samples from each sequence. The resulting dataset contained thousands of training samples across all sequence types, providing diversity.

Model Architecture

We employed a transformer encoder as the predictive architecture model, augmented with several mechanisms for learning numerical sequences. The model accepts a fixed-length window of $L = 30$ normalized sequence values. Each input sequence is presented as a tensor $X \in R^{B \times L}$ where B is the batch size. Because the transformer expects vector-valued inputs, the scalar inputs are first expanded to include a feature dimension and projected into a d_{model} -dimensional embedding space $E_{in} = XW$ where $W \in R^{1 \times d_{model}}$ is a learned linear projection and the embedding dimension is set to $d_{model} = 256$. The model uses learned positional embeddings $P \in R^{L \times d_{model}}$ to provide information about the ordering of sequence elements, which are added to the input embeddings: $E = E_{in} + P$. These positional embeddings are trained jointly with the rest of the network.

Additionally, the model is conditioned on the type of sequence being processed. Each sequence type is represented by a learned embedding vector that is added to every element in the input window. This allows the model to adapt its computations to different structural families without hardcoded mathematical rules.

The embedded sequence is processed by a stack of 6 transformer encoder layers. Each layer employs the following components: multi-head self-attention with 8 attention heads, a position-wise feedforward network with a hidden width of 256, and finally: each sub-layer incorporates a residual connection, dropout with rate 0.1, and layer normalization. The output of the encoder stack is the hidden representation $H \in R^{B \times L \times d_{model}}$.

The prediction is based solely on the hidden state corresponding to the final element of the encoded sequence $h_{last} = H_L$, and a linear output layer maps this vector to a scalar prediction

$\hat{\Delta} = w^T h_{last}$ representing the predicted difference to the next sequence element. The model produces three simultaneous estimates of the next value:

1. Delta prediction: Estimating $\Delta_t = x_{t+1} - x_t$.
2. Ratio prediction: Estimating $r_t = x_{t+1}/x_t$.
3. Direct prediction: Estimating x_{t+1} itself.

The model also produces multiple prediction targets, each generated by its own learned linear output head. To accommodate structural diversity in numerical sequences, the model learns a gating network that outputs non-negative weights summing to one. These weights determine how strongly each prediction strategy influences the final forecast, enabling the model to select the prediction form most suited to the structure present in the sequence.

The model is trained using mean squared error (MSE) loss, minimizing the discrepancy between the predicted delta and the true delta. The model parameters are optimized using the Adam optimizer with a learning rate of 10^{-4}. $L(\theta) = 1/B \sum_{i=1}^{B} (\hat{\Delta}_i - \Delta_i)^2$ where θ represents the model parameters.

To forecast K future steps, the model is applied recursively in an autoregressive manner, repeatedly feeding the model its own predictions. At each step, the model takes the current input window of length L and predicts the next delta. The next sequence value is calculated as $x_{t+1} = x_t + \hat{\Delta}$. The newly predicted value is appended to the sequence and the window is shifted forward one step. The value is denormalized using the sequence's original mean (m) and standard deviation (σ). By iterating this process K times, the model generates the complete forecast $P = \{x_{t+1}, x_{t+2}, \dots, x_{t+K}\}$.

Training Procedure

The model is trained to minimize the prediction error on the next term of each sequence using supervised learning. It predicts normalized targets derived from each sequence rather than training on raw values. For each sequence, both inputs and targets are normalized using the sequence-level mean and standard deviation to ensure stable optimization across sequences of varying scales. Following normalization, the normalized sequences are converted into overlapping training samples using a sliding-window approach with a window length $L = 30$. Each training sample consists of the corresponding targets for the next term and the L-term input window. Three prediction targets are used: additive change, multiplicative change, and the next value itself. Each target is produced by its own output head. Training minimizes the MSE loss across all three prediction targets.

The model is trained strictly in a single-step prediction setting; it only learns to predict the next value given the most recent window. Multi-step forecasting is not performed during training. This separation prevents error accumulation from influencing gradient updates.

The mechanism for multi-step prediction is performed at inference time via recursive autoregression for a forecast horizon of $K = 5$ steps. The loop works as follows: the transformer model takes the current L-length normalized window, X, and produces the three prediction targets along with learned gating weights that determine how to combine them into the next step prediction. The next normalized value is appended to the sequence. Then the window is shifted forward by one step to create the new input window for the next prediction step. The newly generated prediction is denormalized using the original sequence's mean and standard deviation.

This recursive process continues until K steps have been forecast to generate the final multi-step forecast.

To ensure that the model generalizes well to unseen data and does not overfit to specific training sequence patterns, dropout layers within the transformer encoder (rate = 0.1) act as a regularization mechanism.

Results

The performance of the transformer model was assessed using mean squared error (MSE) on the prediction targets and by examining its capacity for multi-step autoregressive forecasting. The model was trained for 10 epochs using a batch size of 256 and the Adam optimizer with a learning rate of 10^{-4}. Training loss and per-sequence-type MSE were computed at every epoch. The tracking of MSE across different sequence types provided relevant insight into the model's performance.

Below is a graph charting the error loss as training progressed, Figure 1.1:

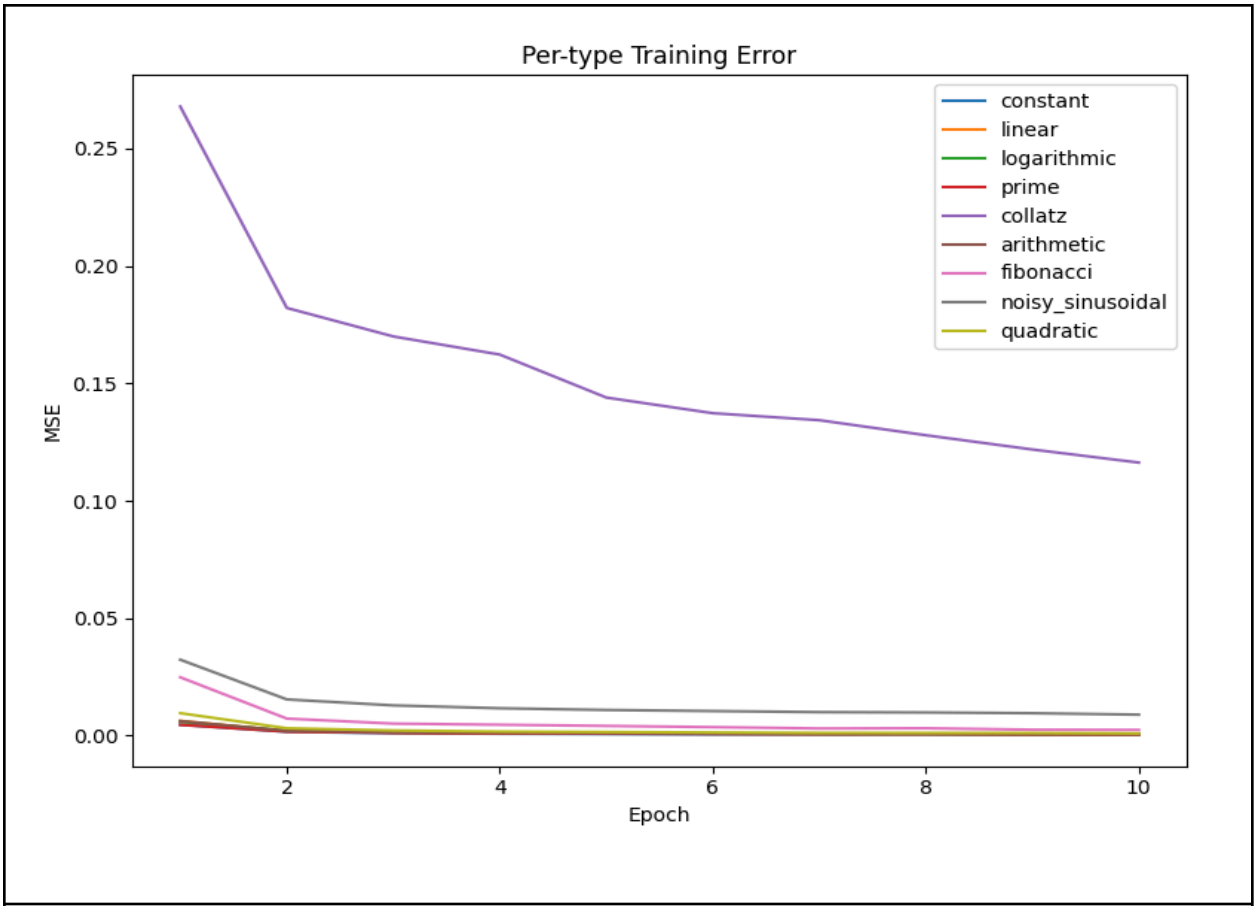


Figure 1.1

Additionally, we ran 50 instead of 10 epochs to see if we could continue to fit the data but the error only decreased minimally for all sequences except for the collatz numbers. This is as expected, and this prolonged training led to overfitting and worse prediction performance. Observe this difference in training error below in Figure 1.2.

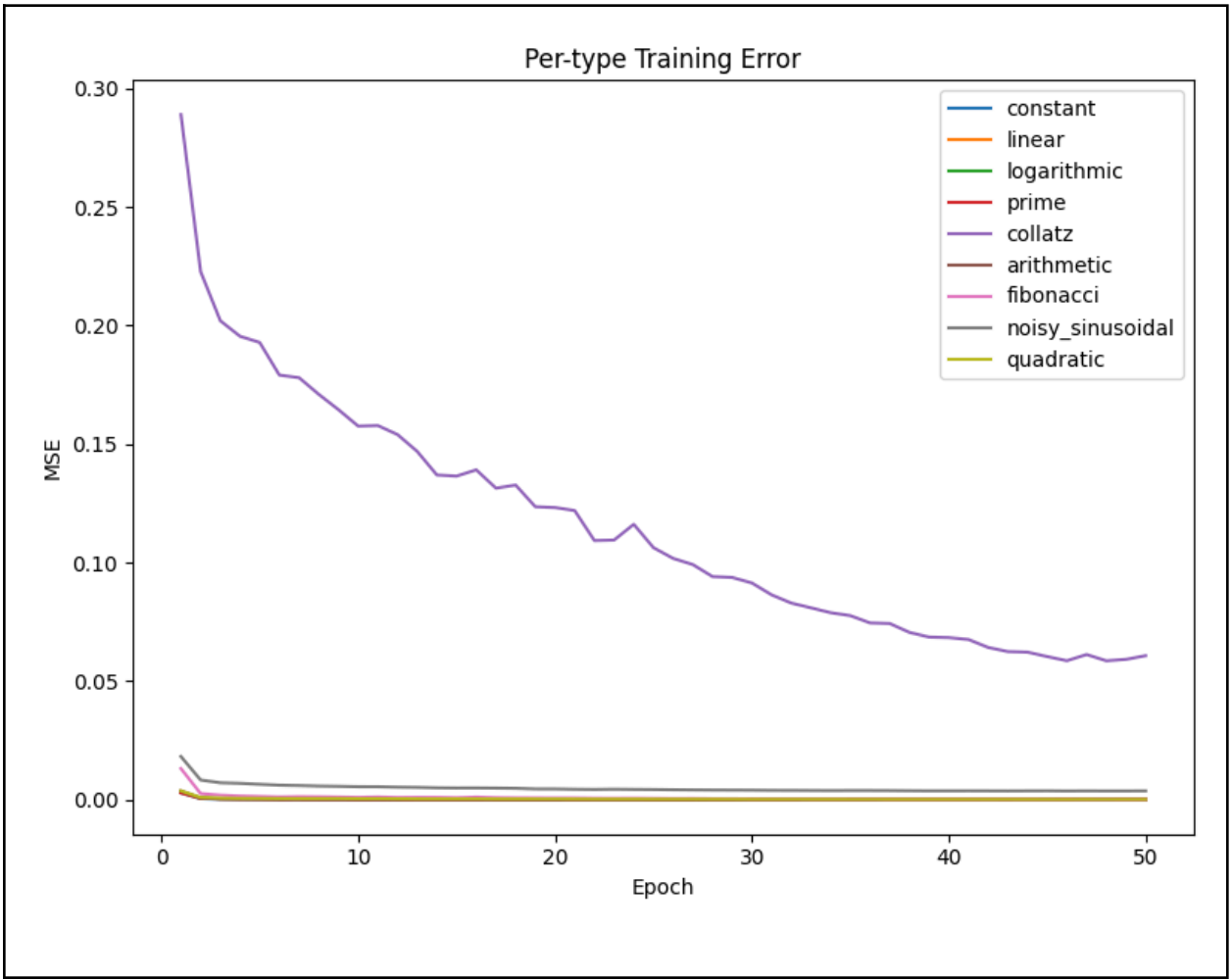
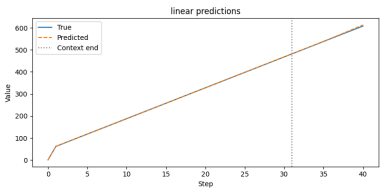
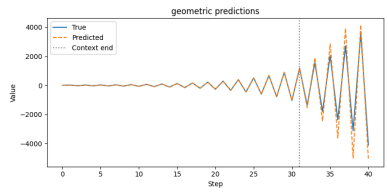
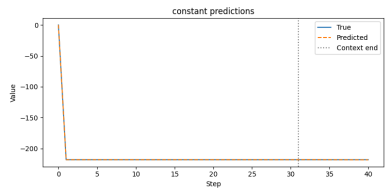
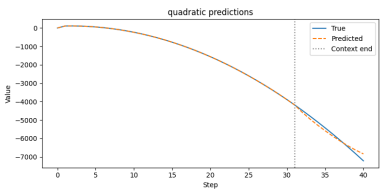
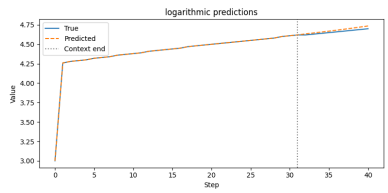
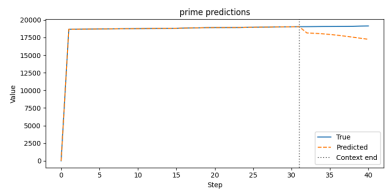
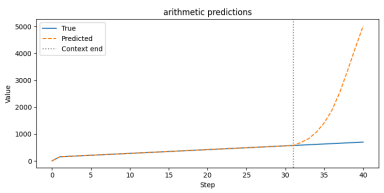
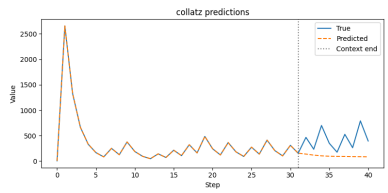
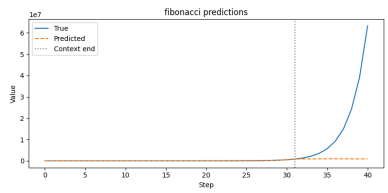
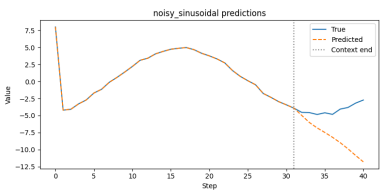

Per-type Training Error
MSE
Epoch
0.30
0.25
0.20
0.15
0.10
0.05
0.00
0
10
20
30
40
50
constant
linear
logarithmic
prime
collatz
arithmetic
fibonacci
noisy_sinusoidal
quadratic

Figure 1.2

We observed that the transformer performed best on constant, linear, geometric, quadratic, and logarithmic sequences: This was as expected, due to their regular structure the model was able to achieve lower error.

		
Linear Predictions MSE: 0.001043	Geometric Predictions MSE: 0.035238	Constant Predictions MSE: 0.000725
		
Quadratic Predictions MSE: 0.001396	Logarithmic Predictions MSE: 0.001128	Prime Predictions MSE: 0.000986
		
Arithmetic Predictions MSE: 0.001040	Collatz Predictions MSE: 0.238690	Fibonacci Predictions MSE: 0.005131
		
Noisy Sinusoidal Predictions MSE: 0.016776		

The following sequence types proved more difficult due to their complex or non-deterministic nature:

Prime number sequences: Prime numbers contain no simple numerical pattern and exhibit irregular gaps.

Fibonacci sequences: Fibonacci numbers require capturing long-range additive structure.

Collatz sequences: Produced the highest error due to complex local structure.

Noisy sinusoidal sequences: Noise disrupts structure.

Arithmetic sequences: Sequences with varying step sizes were sometimes misclassified when delta estimation was unstable.

Multi-step autoregressive forecasting ability was also evaluated. For simple sequence types, the model produced accurate multi-step forecasts. For irregular sequence, error accumulated rapidly which resulted in divergence from the true sequence after a few predicted steps. This behavior is expected given that such sequences lack predictable local structure.

The model demonstrates high accuracy on sequences with simple, predictable delta structures. More complex sequences resulted in a higher per-type MSE. The results confirm that the transformer model is capable of multi-step forecasting, with the forecast reliability being directly correlated to the single-step delta prediction accuracy; unpredictable structures remain challenging.

Limitations

While the transformer model demonstrates solid performance in forecasting some sequences, it exhibits some limitations that should be noted. The transformer model consistently fails to produce accurate forecasts on sequence types with irregular structure, revealing structural limits in its ability to infer mathematical rules and patterns. Most notably, the Collatz numbers presented the most difficulty, since they are very sporadic. Additionally the chaotic sinusoidal is a very chaotic sequence, which despite having an underlying structure, the model faced strong difficulties in predicting these values.

Multi-step predictions at inference time rely on recursive autoregression; each predicted value is used as the input to the next step, and this causes small errors to compound over time. This becomes particularly evident in the Collatz, noisy sinusoidal, and fibonacci sequences.

These limitations suggest that while transformers are capable of learning mathematical sequences, their performance is closely tied to stable structure within the data.

Conclusion

This project investigated the capacity of transformer architectures to learn and predict a wide range of numerical sequence patterns. Our hybrid transformer architecture achieved strong performance in sequences with regular structure. Constant, linear, quadratic, logarithmic, and geometric sequences were learned with high accuracy. These results confirm that transformers are capable of capturing mathematical patterns through self-attention mechanisms without explicitly programming mathematical rules.

However, the model's performance on irregular sequences revealed fundamental limitations. Arithmetic, Fibonacci, noisy sinusoidal, and particular Collatz sequences produced

substantially higher prediction errors. Model performance degraded during multi-step forecasting for these categories, representing a key constraint for the given architecture.

Despite these limitations, our findings show that transformers are capable of successfully modelling certain numerical processes. Future work could explore hybrid symbolic-neural approaches (with explicit procedural logic), or richer embeddings to address the limitations noted. Overall, the insights gained from this project contribute to the understanding of the capabilities and boundaries of transformer models as tools for mathematical sequence learning.