ECAI 2024
U. Endriss et al. (Eds.)

doi:10.3233/FAIA240838

Taylor Expansion in Neural Networks: How Higher Orders Yield Better Predictions

Pavel Zwerschke[a], Arvid Weyrauch[a], Markus Götz[a,b] and Charlotte Debus[a,*]

[a]Scientific Computing Center (SCC),
Karlsruhe Institute of Technology (KIT)
[b]Helmholtz AI
ORCID (Arvid Weyrauch): https://orcid.org/0000-0002-2684-0927, ORCID (Markus Götz): https://orcid.org/0000-0002-2233-1041, ORCID (Charlotte Debus): https://orcid.org/0000-0002-7156-2022

Abstract. Deep learning has become a popular tool for solving complex problems in a variety of domains. Transformers and the attention mechanism have contributed a lot to this success. We hypothesize that the enhanced predictive capabilities of the attention mechanism can be attributed to higher-order terms in the input. Expanding on this idea and taking inspiration from Taylor Series approximation, we introduce "Taylor layers" as higher order polynomial layers for universal function approximation. We evaluate Taylor layers of second and third order on the task of time series forecasting, comparing them to classical linear layers as well as the attention mechanism. Our results on two commonly used datasets demonstrate that higher expansion orders can improve prediction accuracy given the same amount of trainable model weights. Interpreting higher-order terms as a form of token mixing, we further show that second order (quadratic) Taylor layers can efficiently replace canonical dot-product attention, increasing prediction accuracy while reducing computational requirements.

1 Introduction

Time series analysis and forecasting is a vital part of machine learning. The ability to predict temporal behavior is the key to technological innovation, for example in weather and climate forecasting, electricity grid monitoring or autonomous driving. For a long time, statistical approaches such as Kalman-Filters or ARIMA models were the method of choice for analysis and prediction of temporal behavior. However, recent advances in deep learning (DL) have paved the way for neural network-based approaches [6].

The recent hype in artificial intelligence can be largely attributed to the remarkable results produced by the Transformer architecture [16]. Models like GPT-3 [1] have demonstrated not only to the scientific community, but to the broad public, how powerful neural networks can actually be. Recent work on large language models and foundation models has proven that, by learning latent representations of the input data, Transformer-based models are able to generalize across prediction tasks. However, since its introduction, it has not only revolutionized the field of natural language processing, but also time series forecasting. Given their ability to deal with sequential data, Transformer-based models are a natural fit for forecasting uni- and multivariate time series, and numerous model architectures have been proposed [18].

The success of the Transformer architecture is based on its ability to relate individual sequence elements $x \in \mathbb{R}^f$, referred to as tokens, with one another through the self-attention mechanism. Given an input sequence $X \in \mathbb{R}^{f \times L}$ of length L, self-attention provides a mathematical formalism of computing the relationship between all pairs of tokens (token mixing). Various approaches for token mixing have been proposed, e.g., [17, 8], with the canonical dot-product being the most popular one. Apart from the independence of the input sequence length, the dot-product attention mechanism introduces a so far under-recognized key component: while classical neural network layers, like feedforward or convolutions, are *linear* in the input, the dot-product is, from an abstract point of view, *quadratic*. We hypothesize that this quadratic term is the true reason behind the powerful predictive capabilities of the attention mechanism. For that, we interpret neural network layers in the framework of Taylor expansions.

Taylor series approximate arbitrary smooth functions by representing them as a sum of polynomial terms with coefficients calculated from the function's derivatives at a single evaluation point. The more polynomial terms are included, i.e., the higher the order, the better the approximation. Given that Taylor series expansion is typically used for function approximation and that a univariate time series is simply a one-dimensional function, this approach is particularly interesting for the task of time series forecasting.

Hence, we aim to investigate whether we can use higher-order Taylor approximations to improve the degree of function approximation in a neural network. We test this hypothesis by introducing Taylor layers, i.e., representing neural network layers as higher-order polynomials, where the learnable weights can be interpreted as approximations of the underlying function's derivative.

Based on this abstract concept, we further introduce Taylor Transformers, where the classical dot-product attention mechanism is replaced by a Taylor layer of 2$^{\text{nd}}$ order. We evaluate the predictive capabilities of Taylor layers and the Taylor Transformer for time series forecasting in an experimental setup, by evaluating networks with Taylor layers of 2$^{\text{nd}}$ and 3$^{\text{rd}}$ order, i.e., by including quadratic and cubic terms, against classical linear feedforward layers and the canonical dot-product Transformer.

* Corresponding Author. Email: charlotte.debus@kit.edu.

In summary, our contributions include:

- A theoretical interpretation of the dot-product attention mechanism in the light of Taylor series approximation, thus yielding an intuitive understanding of its predictive superiority over other layer architectures;
- A continuation of this thought into the development of Taylor layers, i.e. feedforward neural network layers of higher order, thereby omitting the need for a non-linear activation function while improving function approximation;
- An experimental evaluation of our hypothesis and the introduced Taylor layer architecture in the context of time series forecasting.

Our results demonstrate that with the same number of model weights, Taylor layers yield better function approximations—and thus, forecasting results—than their feedforward counterparts. By evaluating Taylor Transformers against the classical Transformer, we show that our hypothesis to interpret dot-product attention as a higher order Taylor approximation holds empirically, and that proper Taylor expansion in the layer can actually improve prediction results.

2 Related work

2.1 Transformers for Time Series Forecasting

Due to the sequential nature of the data, time series forecasting is a natural fit for Transformer architectures. Hence, it is not surprising that numerous Transformer-based models for time series forecasting have been proposed in the last years. For a comprehensive review, see Wen et al. [18].

However, a common obstacle is the computational complexity of $\mathcal{O}(L^2)$ in the sequence length L, making long sequence prediction particularly challenging. Other works on Transformer-based time series forecasting thus focus on reducing computational complexity in various ways. LogTrans [7] substitutes the point-wise dot-product with causal convolutions and introduces a sparse bias to reduce computational complexity. Informer [23] uses random subsampling of attention queries for dimensionality reduction. Autoformer [21] introduces a local mean-based decomposition method and replaces the dot product attention with an auto-correlation mechanism based on Fourier transforms for lower complexity. This idea is spun further by the FEDformer [24], which selects a fixed number of Fourier modes for auto-correlation. It further employs a decomposition scheme with multiple filter lengths. Pyraformer [9] proposes a pyramidal attention module that uses inter-scale tree structures and intra-scale neighboring connections to leverage multi-resolution representations of time series. Crossformer [22] leverages cross-dimension dependencies for multivariate time series through dimension-segment-wise embedding and a two-stage attention layer. The ETSFormer [20] exploits exponential smoothing attention and frequency attention to replace the self-attention mechanism in vanilla Transformers, thus improving both accuracy and efficiency. In PatchTST [12], the time series is segmented into subseries-level patches which are used as input tokens. Recently, inverted Transformers (iTransformer) [10] were proposed for multivariate time series. The key idea is to embed variables from different series at each time point into variate tokens which are utilized by the attention mechanism to capture multivariate correlations. A feedforward network is used to learn non-linear representations for each variate token. Recent work has aimed to reduce computational complexity via rearranging the input sequence for periodical data [19]. The proposed method ReCycle can be applied in different architectures to achieve a notable reduction in training time and energy consumption.

2.2 Polynomial Networks

Despite the obvious higher predictive capabilities of higher-order polynomials, research for neural networks in this direction has been sparse. This is likely due to the fact that the computational demand of fitting polynomials grows exponentially. Chrysos et al. [3] proposed Π-Nets, a new class of neural networks based on polynomial expansions. The approach implements the neural networks using high-order tensors and estimates the coefficients through factor sharing. Tong et al. [14] construct a three-layer feedforward neural network which uses Taylor series as the activation function of the network. The network is evaluated on a polynomial fitting task for synthetic data. Kileel et al. [5] define the notion of polynomial networks as neural networks that use polynomial activation functions and evaluate their expressiveness.

Multiplicative interactions as a "unifying framework to describe a range of classical and modern neural network architectural motifs" have been studied by [4]. In their study, the authors argue that attention systems like the Transformer use multiplicative interactions to scale different parts of the input sequence.

3 Theory

3.1 Transformers and Dot-Product Attention

Transformers are neural networks that belong to the class of sequence-to-sequence models. As such, they consist of an encoder and a decoder. The encoder processes the input sequence, transforming it into a latent representation. The decoder combines this latent representation with previous outputs and/or decoder inputs to generate an output sequence.

At the heart of both the encoder and the decoder lies the self-attention mechanism, which relates individual sequence elements with one another. We denote $X = \{\boldsymbol{x}_1, ..., \boldsymbol{x}_L\} \in \mathbb{R}^{L \times d_{\text{in}}}$ as a sequence of length L, where the tokens $\boldsymbol{x}_i = \{x_1, ..., x_{d_{\text{in}}}\} \in \mathbb{R}^{d_{\text{in}}}$ are of dimension d_{in} (features). In self-attention, each token is embedded into a *query* $\boldsymbol{q} \in \mathbb{R}^{d_q}$, a *key* $\boldsymbol{k} \in \mathbb{R}^{d_k}$ and a *value* $\boldsymbol{v} \in \mathbb{R}^{d_v}$ via three linear layers. The query and key must be of the same dimension going forward; hence, we abbreviate $d := d_q = d_k$. The three linear embedding layers are defined via corresponding weight matrices $W^q \in \mathbb{R}^{d_{\text{in}} \times d}$, $W^k \in \mathbb{R}^{d_{\text{in}} \times d}$ and $W^v \in \mathbb{R}^{d_{\text{in}} \times d_v}$. Note that these learnable weight matrices are independent of the sequence length L, which is one of the remarkable features of self-attention. However, in practice, tokens are not embedded individually in a sequential manner, but in batches. Hence, we denote the three matrices Q, K, and V as:

$$Q = X \cdot W^q \in \mathbb{R}^{L \times d} \tag{1}$$

$$K = X \cdot W^k \in \mathbb{R}^{L \times d} \tag{2}$$

$$V = X \cdot W^v \in \mathbb{R}^{L \times d_v} \tag{3}$$

where X is a matrix with L column vectors $\boldsymbol{x}_i$. Using the query and key, the *attention matrix* $A \in \mathbb{R}^{L \times L}$ is then computed via

$$\mathrm{A} = \operatorname{softmax}\left(\frac{Q \cdot K^\top}{\sqrt{d_k}}\right) \tag{4}$$

$$= \operatorname{softmax}\left(\frac{X \cdot W^q \cdot (W^k)^\top X^\top}{\sqrt{d_k}}\right) \tag{5}$$

The entry A_{ij} is the attention score between query $\boldsymbol{q}_i$ and key $\boldsymbol{k}_j$. The output sequence $Y \in \mathbb{R}^{L \times d_v}$ of the self-attention layer is computed

as the product of the attention matrix A and the value matrix V.

$$Y = A \cdot V = \mathrm{A} \cdot X \cdot W^v \tag{6}$$

The sequence of output tokens $\boldsymbol{y}_i \in \mathbb{R}^{d_v}$ is then concatenated along the sequence dimension L, and passed through multiple linear layers to generate the latent representation (encoder) or the prediction output (decoder).

3.2 Taylor Series

Taylor expansion is a mathematical concept used to approximate arbitrary smooth functions. It is often used as a vehicle for mathematical proofs but also for the approximation of complex real-world functions. A Taylor series $\mathcal{T}$ is a representation of a function $f(x)$ as an infinite sum of terms that are calculated from the values of the function's nth derivatives $f^{(n)}$ at a single point x_0:

$$\mathcal{T}(x) := \sum_{n=0}^{\infty} \frac{f^{(n)}(x_0)}{n!}(x - x_0)^n \tag{7}$$

The sum of the first k terms of a Taylor series is referred to as a Taylor sum of order k:

$$\mathcal{T}_k(x) := \sum_{n=0}^{k} \frac{f^{(n)}(x_0)}{n!}(x - x_0)^n \tag{8}$$

According to Taylor's theorem [2], the asymptotic behavior of the remainder term is of the same order as the Taylor sum.

$$R_k(x) := f(x) - \mathcal{T}_k(x) = \mathcal{O}(|x - x_0|^k) \tag{9}$$

The concept can be generalized to multi-dimensional functions [11]. Let $\alpha \in \mathbb{N}_0^{d_{\text{in}}}$, $\boldsymbol{x} \in \mathbb{R}^{d_{\text{in}}}$, we define the multi-index notation

$$\begin{aligned} |\alpha| &:= \alpha_1 + \ldots + \alpha_{d_{\text{in}}}\,, \\ \alpha! &:= \alpha_1! \cdots \alpha_{d_{\text{in}}}!\,, \\ \boldsymbol{x}^\alpha &:= x_1^{\alpha_1} \cdots x_{d_{\text{in}}}^{\alpha_{d_{\text{in}}}}\,. \end{aligned} \tag{10}$$

Based on this, we can formulate the k-th order partial derivative D^α, $|\alpha| \leq k$ of $f : \mathbb{R}^{d_{\text{in}}} \to \mathbb{R}$ as

$$D^\alpha f = \frac{\partial^{|\alpha|} f}{\partial x_1^{\alpha_1} \cdots \partial x_n^{\alpha_n}} \tag{11}$$

Using this notation, we can define the multivariate Taylor series of f at $\boldsymbol{x}_0$ as

$$\boldsymbol{\mathcal{T}}(\boldsymbol{x}) := \sum_{k=0}^{\infty} \sum_{|\alpha|=k} \frac{D^\alpha f(\boldsymbol{x}_0)}{\alpha!}(\boldsymbol{x} - \boldsymbol{x}_0)^\alpha,$$

where the second sum is over all α with $|\alpha| = k$.

3.3 Time Series Forecasting

Let $\boldsymbol{x}(t) \in \mathbb{R}^{d_{\text{in}}}$ be a time-dependent, continuous variable with d_{in} features at a point in time t. A time series X is a sequence of N measurements of x over a time span T, taken at times $t_0, t_1, ..., t_N$ with a temporal resolution of $\Delta t = T/N$. $\mathcal{H}$ is the *historic window length* and $\mathcal{F}$ is the *forecast window length*. We define the task of time series forecasting as finding a mapping $\mathcal{M}$, such that

$$\mathcal{M}\left(\boldsymbol{x}(t_{i-\mathcal{H}}), \ldots, \boldsymbol{x}(t_{i-1}), \boldsymbol{x}(t_i)\right) \to \boldsymbol{x}(t_{i+1}, \ldots, t_{i+\mathcal{F}}) \tag{12}$$

for every $\boldsymbol{x}(t_i) \in X$. We align this with the notation introduced in Section 3.1 by abbreviating $\boldsymbol{x}(t_i) := \boldsymbol{x}_i$ and identifying $L = \mathcal{H}$.

4 Taylor layers

A classical feedforward layer in a neural network is defined as a function f that takes the output of the previous layer as input $\boldsymbol{x} \in \mathbb{R}^{d_{\text{in}}}$ and produces an output $\boldsymbol{y} \in \mathbb{R}^{d_{\text{out}}}$. The function is usually an affine-linear transformation $f : \mathbb{R}^{d_{\text{in}}} \to \mathbb{R}^{d_{\text{out}}}$, followed by a non-linear activation function σ:

$$\boldsymbol{y} := f(\boldsymbol{x}) = \sigma(W \cdot \boldsymbol{x} + b)$$

The non-linear activation function σ is essential to the learning capabilities of a neural network, as it allows the network to learn non-linear functions. There exist a variety of activation functions, e.g., the rectified linear unit (ReLU) $\sigma(x) := \max(0, x)$ being one of the most popular.

Without the activation function though, we can interpret such a feedforward layer as a Taylor approximation of first order, with the weights $W_i \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$ merely being a learned approximation of the derivative $D^\alpha f$. Expanding upon this idea, let us have a look at the attention mechanism. The attention matrix A is a bilinear form of the input X with itself with a learned weight matrix

$$W^Q \cdot (W^K)^\top =: (w_{ij})_{ij} \in \mathbb{R}^{d_{\text{in}} \times d_{\text{in}}}\,. \tag{13}$$

Thus, we can rewrite Equation (4) as:

$$\mathrm{A}_{ij} = \sum_{k,m=1}^{d_{\text{in}}} w_{km}(\boldsymbol{x}_i)_k(\boldsymbol{x}_j)_m \quad i, j = 1, \ldots, \mathcal{H} \tag{14}$$

Thus, the attention matrix consists of the inner products of the input X with a learned weight matrix, leading to a quadratic term.

Considering that attention layers are more expressive than linear feedforward layers and hypothesizing that this originates from the higher-order quadratic term. For the full quadratic approach we can ingore the temporal structure of the data and rewrite the input

$$X \in \mathbb{R}^{\mathcal{H} \times d_{\text{in}}} \to \boldsymbol{x} = \{x_1, ..., x_{d_{\text{tot}}}\}\,, \tag{15}$$

where $d_{\text{tot}} = \mathcal{H} \times d_{\text{in}}$. We define a *quadratic layer* component-wise as

$$y_k = \sum_{i,j=1}^{d_{\text{tot}}} \tilde{a}_{ijk} x_i x_j + \sum_{i=1}^{d_{\text{tot}}} a_{ik} x_i + b_k\,, \quad k = 1, \ldots, \mathcal{F} \times d_{\text{out}}\,. \tag{16}$$

Note that an activation function is not necessarily needed anymore, since the layer is already non-linear in the inputs x_i. Since $x_i x_j$ is symmetric, the matrix of coefficients $\tilde{a}_{ijk}$, becomes an upper triangular matrix. Thus, a quadratic layer has

$$\mathcal{F} \times d_{\text{out}} \times \left(\frac{{d_{\text{tot}}}^2 + d_{\text{tot}}}{2} + 1\right) \tag{17}$$

free parameters.

Expanding upon the above idea that a linear layer is a first-order Taylor approximation with learned function derivatives, we interpret the quadratic layer as a second order Taylor sum and hypothesize that it provides better approximations than the linear layer. We then generalize to a *Taylor layer* of order n that approximates the output y generated by an input x by using the first n terms of the Taylor series. Using the multi-index notation for $\alpha \in \mathbb{N}_0^{d_{\text{tot}}}$ from Equation (10), we can write the Taylor layer of order n as the sum over all Taylor terms up to order n of the input features x_i:

$$y_k = \sum_{|\alpha| \leq n} a_{k\alpha} \prod_{i=1}^{d_{\text{tot}}} x_i^{\alpha_i}. \tag{18}$$

The trainable parameters of this layer are all $a_{k\alpha}$, $|\alpha| \leq n$. Taylor layers can be combined to form a Taylor network by simply chaining them together, just like with other layer types.

5 Experimental Evaluation

To test our theory that linear layers and attention layers can be generalized to first- and second-order Taylor approximations, and that higher orders provide better expressiveness, we evaluate the introduced Taylor layers experimentally. Towards this end, we define a set of model architectures that we train and validate on the task of time series forecasting. We train these models on two separate datasets to predict the next $\mathcal{F}$ time steps given the previous $\mathcal{H}$ time steps. We denote this sequence tuple as $(\mathcal{H} \rightarrow \mathcal{F})$. Figure 1 shows an exemplary prediction for forecasting the next 24 time steps, given the last 12 time steps ($\mathcal{H} = 12 \rightarrow \mathcal{F} = 24$).

5.1 Models

For one, we include Taylor networks of second and third order, i.e., employing quadratic and cubic layers, and evaluate them against classical feedforward networks, consisting of linear layers with a non-linear activation function. Note that, unlike in Transformer architectures, the number of parameters in the network is not independent of the sequence lengths $L = \mathcal{H}$, for all three of these models.

The feedforward network and the 2nd-order Taylor network consists of two quadratic layers: one mapping input to hidden layer and one mapping hidden layer to output. For the 2nd-order Taylor network, the size of the hidden layer was set to $h = 10$. Given different input and output sequence lengths, this will result in different numbers of parameters for the networks. To compare the actual expressiveness of higher-order Taylor layers, the number of hidden nodes of the linear model with ReLU activation function was chosen to be roughly the same as the number of parameters of the quadratic model. This is meant to exclude the effect that networks with more parameters typically yield better fits and allows for a fairer comparison of the linear and the quadratic model.

The 3rd-order Taylor network consists of a single cubic layer. We chose to use only a single layer since for cubic layers—the size of the layer in terms of number of parameters is already larger than in the quadratic or linear layers by orders of magnitude.

We further include a classical Transformer model into our evaluation, which consists of a single encoder and decoder layer with one self-attention layer and one feedforward layer each. The parameters d_q, d_k, and d_v of the self-attention layer are set to 16, and the feedforward dimension of the encoder and decoder is 200.

To test our theory that the dot-product attention mechanism can be seen as a bilinear form with a learned weight matrix of the input x with itself, and that it can be generalized to a Taylor approximation of 2nd order, we further included a Taylor Transformer. This model follows the same architecture as the classical Transformer but replaces dot-product attention with a quadratic layer. Note that this results in a much larger number of trainable parameters in the model.

All models were implemented using the PyTorch framework [13]. All code is publicly available on GitHub[1].

[1] https://github.com/RAI-SCC/ModularTransformer/

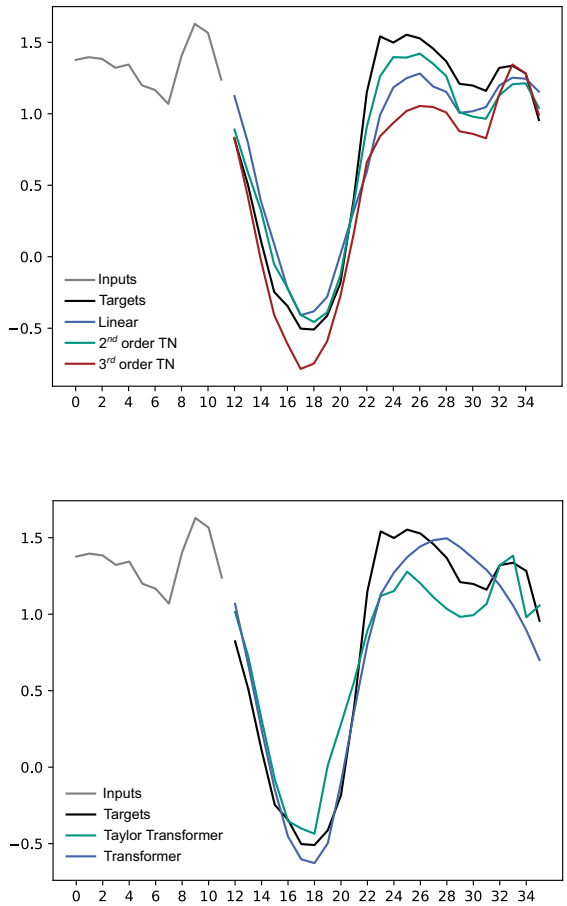

Figure 1. Example predictions of the 2nd- and 3rd-order Taylor network and the classical feedforward network (top) and for the Transformer and Taylor Transformer (bottom) compared to the target output sequence on the ENTSO-E dataset with an (input → output)-sequence length of $(12 \rightarrow 24)$.

5.2 Datasets

We test all models on two datasets commonly used in time series forecasting [23, 24].

The *Western European Power Consumption*[2] dataset (ENTSO-E) is provided by the European Network of Transmission System Operators for Electricity. It contains time-resolved measurements of total electricity consumption of 14 European countries, taken between January 2015 and August 2020 at a temporal resolution of 15 min, 30 min, or 1 h. For our experiments, we use the load data from Germany at a resolution of 1 h. We employ a training–validation split of 70%:30% across the temporal axis.

The *Electricity Transformer Temperature* (ETTh2)[3] dataset contains measurements of power load features and oil temperature of two electricity transformers in China, taken between July 2016 and June 2018 at a temporal resolution of 1 h. In this study, we focus on forecasting the oil temperature of one of the transformers (ETTh2), using a training–validation split of 80%:20% across the temporal axis.

[2] https://www.kaggle.com/datasets/francoisraucent/western-europe-power-consumption

[3] https://github.com/zhouhaoyi/ETDataset

Table 1. Results of fitting the ENTSO-E and ETTh2 dataset with a linear feedforward network, a 2nd-order Taylor network (TN) and a 3rd-order Taylor network, for three different input–output sequence lengths. The table shows mean square error MSE, mean absolute error MAE and total training time for 100 epochs, alongside the number of trainable parameters for each model. Values show the average plus–minus standard deviation over three independent training runs.

	Model	$\mathcal{H}=12 \to \mathcal{F}=24$				$\mathcal{H}=12 \to \mathcal{F}=6$				$\mathcal{H}=24 \to \mathcal{F}=24$			
		MSE	MAPE	Time	#Param	MSE	MAPE	Time	#Param	MSE	MAPE	Time	#Param
ENTSO-E	Linear	0.255 ±0.002	**2.46** ±0.03	341s 24s	2 503	0.088 ±0.002	2.33 ±0.03	304s ±21s	1 317	0.176 ±0.001	2.49 ± 0.05	385s ± 30s	4 875
	2nd-order TN	**0.225** ±0.003	2.93 ±0.09	617s ± 6s	2 494	**0.077** ± 0.001	2.30 ± 0.06	558s ±12s	1 306	0.134 ±0.002	2.22 ±0.02	694s ± 26s	4 834
	3rd-order TN	0.240 ±0.001	2.77 ±0.04	421s ±26s	24 648	0.103 ±0.001	**2.17** ± 0.06	390s ±35s	6 162	**0.128** ±0.002	**2.00** ±0.02	578s ±51s	180 600
ETTh2	Linear	0.146 ±0.001	1.50 ±0.05	115s ±6s	2 503	**0.067** ±0.001	1.02 ± 0.01	99s ±1s	1 317	**0.128** ±0.001	1.57 ±0.08	125s ±4s	4 875
	2nd-order TN	0.147 ±0.002	1.22 ±0.03	217s ±15s	2 494	**0.067** ±0.001	0.97 ±0.02	188s ±3s	1 306	0.13 ±0.006	1.21 ±0.02	241s ±3s	4 834
	3rd-order TN	**0.144** ±0.001	**1.16** ±0.06	146s ±12s	24 648	0.074 ±0.001	**0.96** ±0.01	130s ±4s	6 162	0.129 ±0.002	**1.19** ±0.03	202s ±4s	180 600

5.3 Training

Data was normalized using z-score normalization. All models were trained using Stochastic Gradient Descent (SGD) as an optimizer, with a learning rate of 0.001 and momentum of 0.3 for 100 epochs with a batch size of 32. For training, the Mean Squared Error (MSE) loss was used. Training runs were performed on a MacBook Pro with an M1 Pro chip and 32 GB of RAM. Models were evaluated on the MSE as well as the Mean Absolute Percentage Error (MAPE), which measures the average percentage difference between the actual and predicted values. The MAPE is scale-independent, making it a useful metric for comparing the performance of different models. We further compare total training time and number of parameters of the models. Given that all models were trained for 100 epochs, the total training time directly reflects the time per epoch, and thus, per forward–backward pass.

Training runs were performed three times and the average was taken. We refrain from hyperparameter tuning, because we aim to conduct a fair comparison between models. But, more importantly, we are interested in the relative differences in expressiveness of the different layer types rather than an absolute, superior model performance. As such, we are aware that better training configurations for the models are likely, but are not the focus of these experimental evaluations.

5.4 Results

Table 1 shows results of fitting the two datasets with the 2nd- and 3rd-order Taylor networks and the classical feedforward network, consisting of linear layers with a non-linear activation function. For (input $\to$ output)-sequence lengths of ($12 \to 24$) and ($24 \to 24$), the MAPE decreases consistently with higher layer orders. For ($12 \to 6$), MAPE decreases for the ETTh2 dataset, but increases for ENTSO-E.

These trends can only partially be observed in the MSE: the 2nd-order Taylor network yields lower (ENTSO-E) or on-par (ETTh2) MSE compared to the linear feedforward network, but the 3rd-order network provides increased MSE compared to the 2nd-order Taylor network.

Interestingly, the higher-order Taylor networks yield better predictive performance for (input $\to$ output)-sequence lengths of ($12 \to 24$) and ($24 \to 24$) than for ($12 \to 6$), even though the latter is the easier prediction task. This indicates better extrapolation capabilities of the higher-order Taylor approximations. This finding is further supported by the observation, that the 2nd-order Taylor network almost consistently outperforms the linear feedforward network, even though it has the same number of parameters. The 3rd-order Taylor network on the other hand has up to orders of magnitude more parameters, but yields worse predictions in several cases, indicating a strong tendency to overfit the data.

In terms of computational requirements, the time it takes to complete 100 epochs increases from the linear feedforward network to the 2nd-order Taylor network, which was to be expected. The decrease in training time from the 2nd- to 3rd-order Taylor networks is likely caused by the fact that the 3rd-order Taylor network has one fewer layer. This removes additional computations in the forward–backward pass. Due to the sequentiality of layers, the effect outweighs the increased—but vectorized—computational burden from the cubic term in the input.

We explicitly trained for 100 epochs to gain an estimate of how the training time (per epoch) increases with the increasing computational burden from higher-order terms. However, this does not consider model convergence speeds for the different models. Figure 2 shows exemplary loss curves for all three models, on the task of fitting both datasets with a ($12 \to 24$)-sequence length. We do not observe significantly different convergence behavior for the three models. However, the 2nd- and 3rd-order Taylor networks reach lower loss values earlier, and appear to converge within the 100 epochs, while the linear feedforward network appears to not be fully converged.

Turning to the more complex model architectures, Table 2 lists results for fitting the two datasets with the Taylor Transformer compared to the classical Transformer, that uses dot-product attention.

The Taylor Transformer, i.e., substituting dot-product attention with a simple quadratic layer, yields consistently lower MSE and MAPE, except for ($12 \to 24$)-sequence lengths ENTSO-E MAPE. But, more importantly, it requires less time per epoch, despite having substantially more parameters to fit. This is likely caused by the fact that the dot-product attention mechanism requires multiple, sequentially performed matrix multiplications in the forward and backward passes. To compute the attention matrix, the Transformer first needs to compute $Q = W_q \cdot X$, $K = W_k \cdot X$, and $V = W_v \cdot X$, even though these computations can be performed in a batch. After that, the attention matrix is computed via the product of Q and K, and then multiplied with V. The Taylor Transformer, on the other hand, only needs to compute the quadratic Taylor layer which comprises

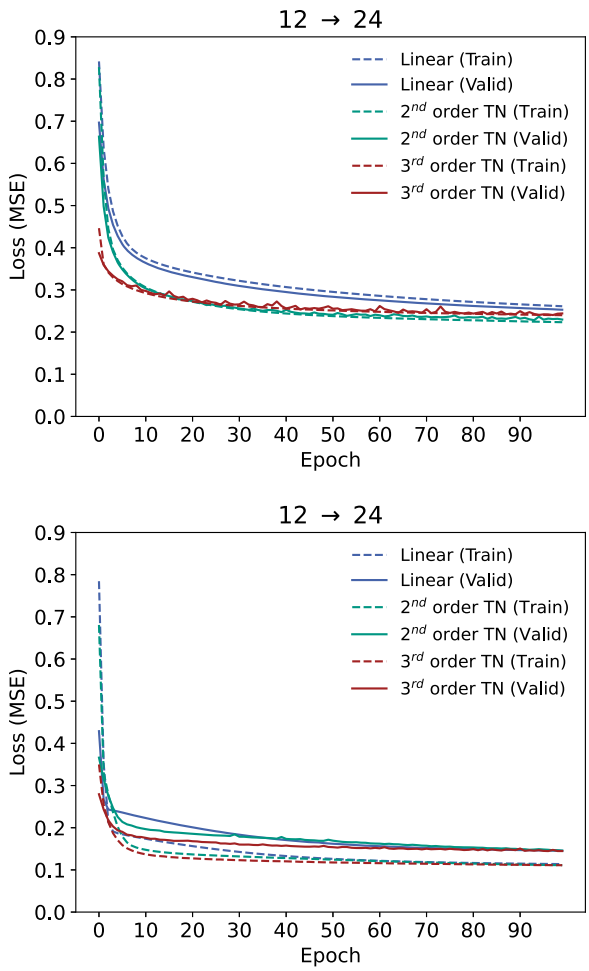

Figure 2. Training and validation loss for the ENTSO-E dataset (top) and the ETTh2 dataset (bottom) with an (input → output)-sequence length of (12 → 24). The curves show the MSE loss for the 2nd- and 3rd-order Taylor network and the classical feedforward network.

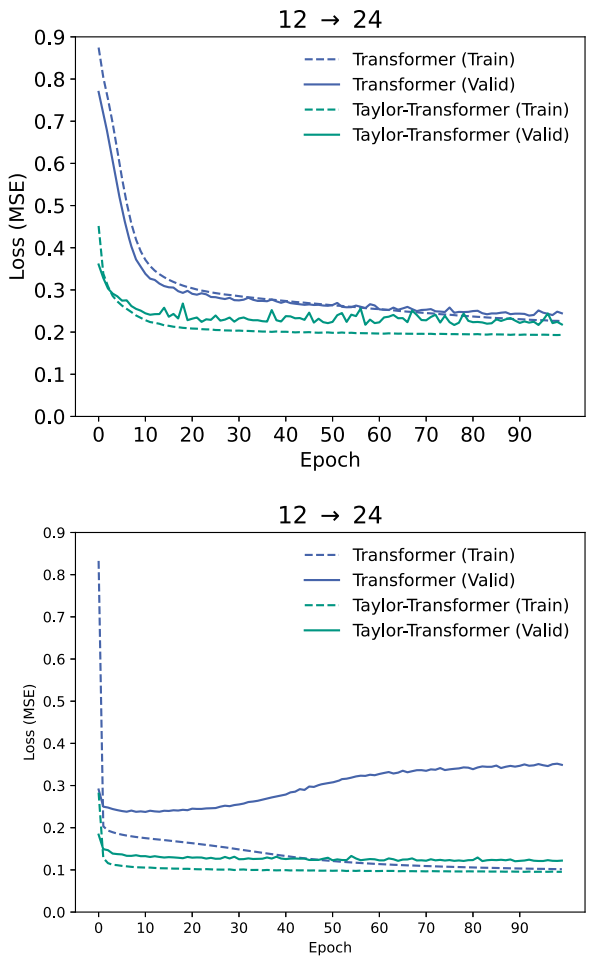

Figure 3. Training and validation loss for the ENTSO-E dataset (top) and the ETTh2 dataset (bottom) with an (input → output)-sequence length of (12 → 24). The curves show the MSE loss for the 2nd- and 3rd-order Taylor network and the classical feedforward network.

two matrix multiplications that can be performed simultaneously.

That being said, we observed that on the ETTh2 dataset, the classical Transformer often converged after only a few epochs and even overfit, as shown by the corresponding loss curves in Figure 3.

6 Conclusion

Transformers are without doubt one of the most relevant methodological developments in deep learning in the past decade and have revolutionized the field of time series forecasting. We hypothesized that the immense predictive capabilities result from the quadratic functional dependency in the sequence input, taking analogy from Taylor series expansions. We tested this hypothesis through empirical evaluation, introducing Taylor layers as higher-order neural network layers.

Our experimental results demonstrate the potential of using higher-order Taylor approximations as neural network layers. Even with the same amount of parameters, 2nd-order Taylor layers are able to capture temporal dynamics more accurately. Using quadratic layers in the attention mechanism yields higher predictive performance at a comparable training time, even though the number of parameters is significantly higher. Our study provides a proof-of-concept and, thus, did not focus on achieving state-of-the-art model training. It is obvious that hyperparameter tuning and advanced optimization algorithms can further improve the prediction results.

Our proposed Taylor layers have noticeably more parameters than classical linear layers. In theory, this requires correspondingly more resources. However, this does not necessarily result in longer training times due to more optimal vectorization, as demonstrated by our experiments on Transformer architectures. Furthermore, computational complexity of higher-order Taylor layers grows exponentially with input dimensionality. This can pose a significant challenge bottleneck, as observable in our experiments on 2nd- and 3rd-order Taylor networks. Hence, Taylor layers are less suitable for inputs with high feature dimensionality, such as images. For uni- and multivariate time series, this issue might also arise with longer sequences. Yet, the computational complexity can be attributed solely to large matrix–matrix multiplications in the forward and backward passes. Hence, parallel and distributed matrix multiplication algorithms, such as SUMMA [15], may be used in the future to alleviate this bottleneck. All in all, Taylor layers provide a promising new architecture for neural networks and, in particular, for time series forecasting tasks.

Table 2. Results of fitting the ENTSO-E and ETTh2 dataset with a classical Transformer (Transf.), using dot-product attention and a Taylor Transformer (Taylor T.), for three different input–output sequence lengths. The table shows mean square error MSE, mean absolute error MAE and total training time for 100 epochs, alongside the number of trainable parameters for each model. Values show the average plus–minus standard deviation over three independent training runs.

	Model	$\mathcal{H} = 12 \rightarrow \mathcal{F} = 24$				$\mathcal{H} = 12 \rightarrow \mathcal{F} = 6$				$\mathcal{H} = 24 \rightarrow \mathcal{F} = 24$			
		MSE	MAPE	Time	#Param	MSE	MAPE	Time	#Param	MSE	MAPE	Time	#Param
ENTSO-E	Transf.	0.245 ±0.013	3.27 ±0.09	701s ±73s	3 569	0.108 ±0.011	2.56 ± 0.24	537s ± 133s	3 569	0.238 ±0.02	2.76 ±1.26	837s ± 123s	3 569
	Taylor T.	**0.213** ±0.005	**2.11** ±1.38	699s ±59s	10 424	**0.058** ±0.004	**1.65** ±0.39	574s ±85s	2 558	**0.114** ±0.007	**1.54** ±0.59	825s ±90s	17 420
ETTh2	Transf.	0.239 ±0.001	2.15 ±0.02	282s ±41s	3 569	0.255 ±0.012	2.19 ±0.01	202s ±23s	3 569	0.218 ±0.001	2.06 ±0.04	318s ±22s	3 569
	Taylor T.	**0.129** ±0.008	**1.19** ±0.27	264s ±46s	10 424	**0.052** ±0.005	**0.87** ±0.04	206s ±18s	2 558	**0.123** ±0.018	**1.16** ±0.27	295s ±7s	17 420

Acknowledgements

This work is supported by the Helmholtz Association Initiative and Networking Fund under the Helmholtz AI platform grant and the HAICORE@KIT partition and by the German Federal Ministry of Education and Research under the 01IS22068 - EQUIPE grant. The authors gratefully acknowledge the computing time made available to them on the high-performance computer HoreKa at the NHR Center KIT. This center is jointly supported by the Federal Ministry of Education and Research and the state governments participating in the NHR (www.nhr-verein.de/unsere-partner).

References

[1] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[2] C. Canuto and A. Tabacco. *Mathematical Analysis I*. Springer Cham, 2015. ISBN 9783319127729.

[3] G. G. Chrysos, S. Moschoglou, G. Bouritsas, J. Deng, Y. Panagakis, and S. Zafeiriou. Deep polynomial neural networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(8):4021–4034, 2022. doi: 10.1109/TPAMI.2021.3058891.

[4] S. M. Jayakumar, W. M. Czarnecki, J. Menick, J. Schwarz, J. Rae, S. Osindero, Y. W. Teh, T. Harley, and R. Pascanu. Multiplicative interactions and where to find them. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=ryln K6VtDH.

[5] J. Kileel, M. Trager, and J. Bruna. On the expressive power of deep polynomial neural networks, 2019.

[6] P. Lara-Benitez, M. Carranza-Garcia, and J. C. Riquelme. An experimental review on deep learning architectures for time series forecasting. *International Journal of Neural Systems*, 31(03):2130001, 2021.

[7] S. Li, X. Jin, Y. Xuan, X. Zhou, W. Chen, Y.-X. Wang, and X. Yan. Enhancing the locality and breaking the memory bottleneck of transformer on time series forecasting. *Advances in neural information processing systems*, 32, 2019.

[8] Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, and A. Anandkumar. Fourier neural operator for parametric partial differential equations. *arXiv preprint arXiv:2010.08895*, 2020.

[9] S. Liu, H. Yu, C. Liao, J. Li, W. Lin, A. X. Liu, and S. Dustdar. Pyraformer: Low-complexity pyramidal attention for long-range time series modeling and forecasting. In *International Conference on Learning Representations*, 2021.

[10] Y. Liu, T. Hu, H. Zhang, H. Wu, S. Wang, L. Ma, and M. Long. itransformer: Inverted transformers are effective for time series forecasting. *arXiv preprint arXiv:2310.06625*, 2023.

[11] J. Marsden and A. Tromba. *Vector Calculus*. Macmillan Learning, 2012. ISBN 978-1429224048. URL https://books.google.de/books?id =pVbIygAACAAJ.

[12] Y. Nie, N. H. Nguyen, P. Sinthong, and J. Kalagnanam. A time series is worth 64 words: Long-term forecasting with transformers. *arXiv preprint arXiv:2211.14730*, 2022.

[13] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

[14] Y. Tong, L. Yu, S. Li, L. Jingyi, H. Qin, and L. Weijun. Polynomial fitting algorithm based on neural network. *ASP Transactions on Pattern Recognition and Intelligent Systems*, 1:32–39, 04 2021. doi: 10.52810 /TPRIS.2021.100019.

[15] R. A. Van De Geijn and J. Watts. Summa: Scalable universal matrix multiplication algorithm. *Concurrency: Practice and Experience*, 9(4): 255–274, 1997.

[16] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee24354 7dee91fbd053c1c4a845aa-Paper.pdf.

[17] G. Wei, Z. Zhang, C. Lan, Y. Lu, and Z. Chen. Active token mixer. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 2759–2767, 2023.

[18] Q. Wen, T. Zhou, C. Zhang, W. Chen, Z. Ma, J. Yan, and L. Sun. Transformers in time series: A survey. *arXiv preprint arXiv:2202.07125*, 2022.

[19] A. Weyrauch, T. Steens, O. Taubert, B. Hanke, A. Eqbal, E. Götz, A. Streit, M. Götz, and C. Debus. Recycle: Fast and efficient long time series forecasting with residual cyclic transformers. In *2024 IEEE Conference on Artificial Intelligence (CAI)*, pages 1187–1194, 2024. doi: 10.1109/CAI59869.2024.00212.

[20] G. Woo, C. Liu, D. Sahoo, A. Kumar, and S. Hoi. Etsformer: Exponential smoothing transformers for time-series forecasting. *arXiv preprint arXiv:2202.01381*, 2022.

[21] H. Wu, J. Xu, J. Wang, and M. Long. Autoformer: Decomposition transformers with auto-correlation for long-term series forecasting. *Advances in Neural Information Processing Systems*, 34:22419–22430, 2021.

[22] Y. Zhang and J. Yan. Crossformer: Transformer utilizing cross-dimension dependency for multivariate time series forecasting. In *The Eleventh International Conference on Learning Representations*, 2022.

[23] H. Zhou, S. Zhang, J. Peng, S. Zhang, J. Li, H. Xiong, and W. Zhang. Informer: Beyond efficient transformer for long sequence time-series forecasting. *arXiv preprint arXiv:2012.07436*, 2021.

[24] T. Zhou, Z. Ma, Q. Wen, X. Wang, L. Sun, and R. Jin. FEDformer: Frequency enhanced decomposed transformer for long-term series forecasting. *arXiv preprint arXiv:2201.12740*, 2022. doi: 10.48550/arXiv .2201.12740. URL http://arxiv.org/abs/2201.12740. version: 3.