(MPO)²: Multivariate Polynomial Optimization based on Matrix Product Operators

Niccolo' Ciolli [1] **Anders Vestergaard Nørskov** [1] **Michael Kastoryano** [2 3] **Petr Taborsky** [1] **Morten Mørup** [1]

Abstract

Central to machine learning is the ability to perform universal function approximation and learn complex input-output relationships from limited numbers of observations. Multivariate polynomial models offer a natural way to express such relationships through multiplicative feature interactions, but their coefficient tensors grow exponentially with the polynomial degree. Existing tensorized polynomial models reduce this cost, yet canonical polyadic decompositions have rank-limited expressivity and existing tensor train formulations are feature order dependent. We introduce $(\mathrm{MPO})^2$, a multivariate polynomial optimization framework based on matrix product operators that combines learned MPO feature embeddings with compact MPO polynomial weight tensors. This yields feature order independent polynomial representations that can incorporate structured operators such as projections, convolutions, and masks for weight tensor symmetries. Across regression and classification benchmarks, $(\mathrm{MPO})^2$ improves over existing tensor decomposition based polynomial models and provides a flexible alternative for efficient polynomial function approximation.

[1]DTU Compute Cognitive Systems, Technical University of Denmark (DTU), Copenhagen, Denmark [2]Department of Computer Science Quantum algorithms and tensor networks, University of Copenhagen (KU), Copenhagen, Denmark [3]AWS Center for Quantum Computing. Correspondence to: Niccolo' Ciolli <nicci@dtu.dk>.

Proceedings of the 43rd International Conference on Machine Learning, Seoul, South Korea. PMLR 306, 2026.
Copyright 2026 by the author(s).

1. Introduction

A central task in machine learning is to learn suitable functions from data that can map inputs to associated outputs in a way that generalizes in the best possible way. Whereas it is well established that deep learning can provide universal function approximation for sufficiently large model architectures (Hornik et al., 1989), many other modeling tools exist for universal function approximations. This includes Gaussian Processes (GPs) for suitable choices of kernels (Williams & Rasmussen, 2006; Tran et al., 2016) and polynomial regressions with universal approximations guaranteed by Taylor's theorem.

In the recent decade context aware learning methods have demonstrated superior performance in generalization by enabling to leverage non-linear dependencies. This includes the transformer architecture (Vaswani et al., 2017) and gating mechanisms as used for instance in long-short term memory (Hochreiter & Schmidhuber, 1997) and gated linear units (Dauphin et al., 2017). Importantly, such architectures can directly leverage multiplicative interactions between attributes instrumental in deep learning (Jayakumar et al., 2020). Whereas multiplicative interactions are well established in classical statistics as defined by widely used interaction terms between features, the ability to directly account for simple multiplication operations within a standard multilayer feed forward network has been shown to require several (i.e., four) hidden neurons (Lin et al., 2017).

Conversely, polynomial networks directly express multiplicative interactions using higher degree terms. Early works on polynomial networks were based on the pi-sigma network (Shin & Ghosh, 1991) that expressed polynomials as multiplications of simple linear regression functions. Ridge polynomial networks (Shin & Ghosh, 1995) similarly express the polynomial function in terms of successive accumulations whereas the pi-sigma-pi network introduced an additional multiplicative layer combining multiple pi-sigma networks (Li, 2003).

Recently, polynomial networks have been advanced to explore tensor decomposition structures including the canonical polyadic decomposition (CPD) (Hendrikx et al., 2019; Govindarajan et al., 2022; Ayvaz & De Lathauwer, 2022; Kilic & Batselier, 2025) as well as hierarchically coupled CPD decompositions forming the P-Net (Chrysos et al., 2022b;a) which are directly related respectively to the pi-sigma and ridge polynomial networks that can be considered rank one CPD structured. Besides the CPD, these approaches have also been advanced to more flexible tensor network structures including the tensor train/matrix product states decomposition (MPS/TT) (Stoudenmire & Schwab, 2016a; Götte et al., 2021; Kilic & Batselier, 2025). Importantly, decomposed polynomial networks can be optimized using simple alternating linear systems (ALS) optimization using second order methods to optimize each factor of the decomposition at a time, see also Hendrikx et al. (2019); Ayvaz & De Lathauwer (2022); Kilic & Batselier (2025). Whereas the above polynomial networks explore tensor decompositions for regression we note that they differ from tensor regression which aims to explore regression of high order data structures (Liu et al., 2022). Tensor network representation for polynomial networks also differ from recent efforts using tensor decomposition procedures to compress the weight tensors in deep learning models, for a discussion of the connections between tensor decompositions and deep learning see also Panagakis et al. (2021).

Importantly, tensor decomposition structures address the curse of dimensionality of the multivariate polynomial regression weights (Shin & Ghosh, 1991). However, the existing formulations using the CPD decomposition have limited expressive power, whereas the current MPS/TT based modeling procedures are feature order dependent. Furthermore, these procedures do not account for redundancies in the weight tensor and are based on prespecified feature representations. These limitations, we argue, have hampered the wider adoption of this otherwise attractive alternative to deep learning based function approximation.

We presently propose the Multivariate Polynomial Optimization based on Matrix Product Operators $(\mathrm{MPO})^2$ framework, a new tensor network based structure for the modeling of higher order polynomials. Notably, $(\mathrm{MPO})^2$ generalizes polynomial tensor networks enhancing their

- **Expressiveness:** We consider more expressive tensor network representations exploring the matrix product operators formalism to both learn feature and polynomial representations with added expressive capabilities when compared to CPD and existing MPS/TT based procedures notably also being feature order independent when compared to the latter.
- **Versatility:** We introduce generic structured operators to account for inductive biases such as polynomial degree redundancies and translation invariance as imposed by conventional convolutional neural networks. We further accommodate different loss functions such as least squares for regression and cross-entropy minimization for classification using a loss-agnostic second order minimization framework.

We evaluate the proposed $(\mathrm{MPO})^2$ structure for supervised learning on several tabular datasets and on image classification, and highlight its advantages over the latest tensor network based methods.

2. Methods

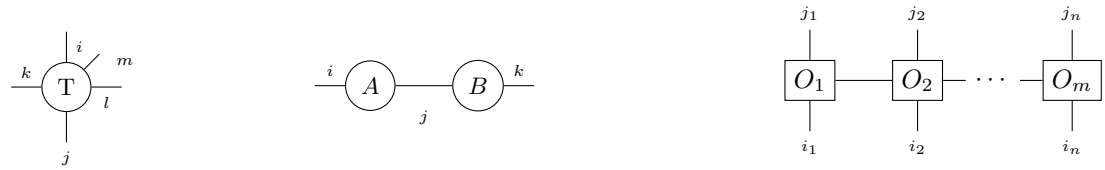
2.1. Tensor Networks and Tensor Notation

Tensor networks are represented by graphs where each node is a tensor and the connections represent contraction over a mode of the tensor. A trivial example of graph representations of tensor networks can be seen in 1

In this work, the position of the indices of a tensor, when referring to tensor networks structures will be at the superscript to indicate vertical modes in the diagrammatic representation and at the subscript to indicate horizontal modes. The different positions have no mathematical difference, but carries a conceptual difference where vertical modes will be associated to the input space while horizontal modes will be associated with the latent space. Summation over multiple indices of the tensor M with elements $\mathrm{M}_{i_1 i_2 \ldots i_n}$ will be denoted by $\sum_{i_1 i_2 \ldots i_n} \mathrm{M}_{i_1 i_2 \ldots i_n} = \sum_{\boldsymbol{i}} \mathrm{M}_{\boldsymbol{i}^{(n)}}$, where $\boldsymbol{i}^{(n)} = \{i_1, i_2, \ldots, i_n\}$, meaning that the sum is performed over all the indices going from i_1 to i_n. When the superscript is omitted, it means that $\boldsymbol{i} = \boldsymbol{i}^{(N)}$, where N is the degree of the polynomial.

A trivial example of a graphical representation of tensor networks can be seen in Fig. 1 in which, in the left panel, a tensor with five modes is illustrated, and in the middle panel a contraction of two tensors multiplied along one mode corresponding to conventional matrix multiplication, and in the right panel the matrix product operator (MPO) corresponding to multiple tensors being pairwise contracted along one mode.

The well-known tensor network structures, namely *ma-*



(a) 5-d tensor T_{ijklm}

(b) Matrix product: $T_{ik} = \sum_j A_{ij} B_{jk}$

(c) MPO: $\mathrm{O}^{\boldsymbol{i}^N \boldsymbol{i}'^N} = \sum_{\boldsymbol{a}^{(n)}} A^{[1]i_1 i'_1}_{1a_2} A^{[2]i_2 i'_2}_{a_2 a_3} \dots A^{[n]i_n i'_n}_{a_n 1}$

Figure 1. Graphical representation of (a) tensors, (b) the matrix product, and (c) the matrix product operator (MPO).

trix product states (*MPS*) and *tensor trains* (*TT*), as well as Tucker and CPD decompositions, are respectively given by

$$\begin{aligned} \text{MPS/TT:}\, \mathrm{T}_{l\boldsymbol{d}^{(N)}} &= \sum_{\boldsymbol{r}}^{R} \mathrm{T}^{[1]d_1}_{1r_2} \mathrm{T}^{[2]d_2}_{r_2 r_3} \dots \mathrm{T}^{[n]d_n l}_{r_n 1} \\ \text{Tucker/CPD:}\, \mathrm{T}_{l\boldsymbol{d}^{(N)}} &= \sum_{\boldsymbol{r}}^{R} \mathcal{G}_{\boldsymbol{r}} \mathrm{T}^{[1]d_1}_{r_1} \dots \mathrm{T}^{[n]d_n l}_{r_n}, \end{aligned} \tag{1}$$

in which the Tucker decomposition reduces to the CPD when $\mathcal{G} = \mathcal{I}$, i.e., is defined as the identity tensor with ones along the (hyper-)diagonal and zeros elsewhere. Notably, these decompositions are special cases of MPOs.

2.2. Multivariate Polynomial Regression

Given an input $\boldsymbol{x}$ of dimension D, we define a multivariate polynomial of $\boldsymbol{x}$ of degree N:

$$\begin{aligned} p_l(\boldsymbol{x}) = \mathrm{T}^{(0)}_l + \sum_{d_1} \mathrm{T}^{(1)}_{ld_1} x_{d_1} + \sum_{d_2 \geq d_1} \mathrm{T}^{(2)}_{l\boldsymbol{d}^{(2)}} x_{d_1} x_{d_2} + \\ + \dots + \sum_{d_2 \geq d_1} \dots \sum_{d_N \geq d_{N-1}} \mathrm{T}^{(N)}_{l\boldsymbol{d}^{(N)}} x_{d_1} \dots x_{d_N} \end{aligned} \tag{2}$$

where l indicates one of the multivariate polynomial outputs, and T are the coefficients.

Following Ayvaz & De Lathauwer (2022), we consider two formulations of the polynomial. Namely, as a sum of independent homogeneous polynomials of increasing degree with a tensor for each degree specifying all coefficients within this degree (type I), or one tensor to represent all coefficients of the polynomial (type II):

$$\text{Type I:} \quad p_l(\boldsymbol{x}) = \sum_{n=0}^{N} \sum_{\boldsymbol{d}^{(n)}} \mathrm{T}^{(n)}_{l\boldsymbol{d}^{(n)}} x_{d_1} \dots x_{d_n}, \tag{3}$$

$$\text{Type II:} \quad p_l(\tilde{\boldsymbol{x}}) = \sum_{\boldsymbol{d}^{(N)}} \tilde{\mathrm{T}}_{l\boldsymbol{d}^{(N)}} \tilde{x}_{d_1} \dots \tilde{x}_{d_N} \tag{4}$$

where $\tilde{\boldsymbol{x}}$ will be the input vector $\boldsymbol{x}$ with a constant additional feature of value 1 (a bias term) in order to account for all coefficients of all the different degrees in the modeling.

Notably, these weight tensors grow exponentially in the number of coefficients as the degree N of the polynomial increases for M features by $\mathcal{O}(\mathcal{M}^{\mathcal{N}})$ making the polynomial regression considering all coefficients of all degrees infeasible except for low degree polynomials with limited numbers of features. To reduce the number of parameters these weight tensors have been decomposed using the CPD decomposition (Hendrikx et al., 2019; Ayvaz & De Lathauwer, 2022; Govindarajan et al., 2022; Chrysos et al., 2022a; Kilic & Batselier, 2025) as well as tensor train decomposition (Stoudenmire & Schwab, 2016a; Götte et al., 2021; Kilic & Batselier, 2025). However, these existing CPD procedures have limited modeling capacity whereas the existing TT modeling procedures are feature order dependent decomposing the weight tensors using feature specific carts, i.e., $\mathrm{T}^{[m]d'_{m-1}}_{r_{m-1} r_m}$ associates the decomposed weight tensor with feature m, which is undesirable as there typically exists no natural feature ordering of the decomposition. As we will show, these drawbacks can be effectively addressed considering the MPO formalism.

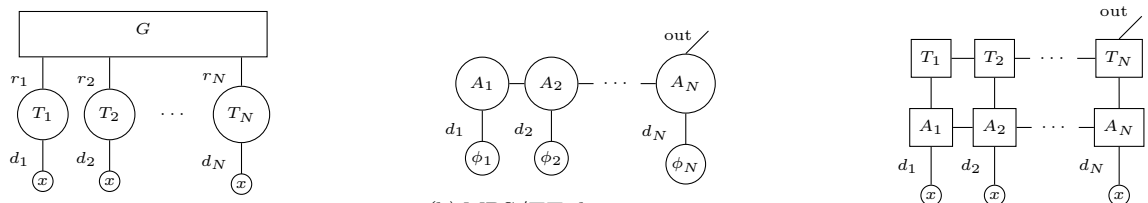
2.2.1. (MPO)^2: Multivariate Polynomial optimization using Matrix Product Operators

Often in machine learning, to enhance the capability of the model, a linear transformation is applied to the inputs to learn suitable latent feature representations. By applying a generic set of transformations $\boldsymbol{A}^{[i]}$ to the inputs, we can express the polynomial as:

$$\begin{aligned} p_l(\boldsymbol{x}) &= \sum_{\boldsymbol{d}\boldsymbol{d}'} \mathrm{T}_{\boldsymbol{d}'l} A^{[1]}_{d'_1 d_1} x_{d_1} \dots A^{[N]}_{d'_N d_N} x_{d_N} \\ &= \sum_{\boldsymbol{d}\boldsymbol{d}'} \mathrm{T}_{\boldsymbol{d}'l} \mathrm{A}_{\boldsymbol{d}'\boldsymbol{d}} x_{d_1} \dots x_{d_N} \end{aligned} \tag{5}$$

where we omit the superscript $\boldsymbol{d}^{(N)}$ when it is equal to N, the degree of the polynomial. We defined the product of all linear operators $\boldsymbol{A}$ as a tensor A.

In the (MPO)^2 framework we propose to perform mul-



(a) Tucker/CPD decomposition *(b)* MPS/TT decomposition *(c)* $(\text{MPO})^2$

Figure 2. Existing tensor network modeling procedures for multivariate polynomial regression based on (a) the CPD decomposition, (b) the MPS/TT decomposition in which each feature has its own train-cart operating on a fixed transformation $\phi(x_i) = \phi_i$ imposed on each feature, and (c) the proposed $(\text{MPO})^2$ framework exploring two layers of MPOs respectively transforming the input to suitable latent representations and creating a feature order invariant polynomial representation.

tivariate polynomial regression and classification by modeling both the generic linear transformation of the input space as well as the polynomial coefficients as *matrix product operators* (MPOs). These are diagrammatically represented in Fig. 2c and given as follows:

$$\mathrm{T}_{\boldsymbol{d}'^{(N)}l} = \sum_{\boldsymbol{r}}^{R} \mathrm{T}^{[1]d'_1}_{1r_2} \mathrm{T}^{[2]d'_2}_{r_2r_3} \dots \mathrm{T}^{[n]d'_n l}_{r_n 1}, \tag{6}$$

$$\mathrm{A}_{\boldsymbol{d}'\boldsymbol{d}} = \sum_{\boldsymbol{r}}^{R} A^{[1]d_1d'_1}_{1r_2} A^{[2]d_2d'_2}_{r_2r_3} \dots A^{[n]d_nd'_n}_{r_n 1} \tag{7}$$

where R is the rank of the MPO structure.

We propose three structures of the MPOs representing the input transformation tensor A by linear projections, convolutions, and masking that accounts for weight redundancies by the proposed masking MPO. These three structures are outlined below, whereas a more comprehensive mathematical description can be found in Appendix A.1.

2.2.2. Linear projection MPO

We project the inputs subspace, related to a single degree, from D to D' dimensional space, where $D' \ll D$, reducing parameters and complexity of inverting Hessian during inference by a factor $\sim \left(\frac{D}{D'}\right)^3$. The operator in its most general form as in Eq. (12) can be randomly initialized and learned blockwise in the same fashion as the train structured coefficients are learned, in such a way that the model learns the best transformation of the inputs. Especially for high-dimensional inputs, computing and inverting the Hessian of a block can be challenging, and, if the inputs show linear dependency, wasteful. We introduce a learnable operator that applies a simple linear transformation for each subspace represented by the blocks, which results in global structured linear transformation. To further reduce the parameters, we can impose independency between the subspaces for the linear transformation by setting the rank to one. The advantage is that the new model, instead of representing the coefficients of the polynomial with blocks of dimension R^2D, instead is represented by two blocks of dimensions R^2D' and DD', where D is the dimension of the input and D' is the dimension of the projected subspace. We define the linear MPO block as a randomly initialized learnable tensor $\mathrm{B}^{i_i i'_i}_{r_i r_{i+1}}$. When the dimension of the rank r is 1 we retrieve linear independent transformations of the inputs.

2.2.3. Convolution MPO

A structured case of linear projection are convolutions, which accounts for translation invariant compression as explored in CNNs (Olshausen & Field, 1996). By representing the inputs as a two dimensional tensor, and projecting the inputs along one of these two dimensions we obtain an operation equivalent to convolution. For images, we can define the two dimensions as patches and pixels in each patch, respectively called p and k in the following. The convolution with kernel g can then be written as $x_p = \sum_k g_k X_{k,p}$. Consequently, the convolution MPO block will have the form:

$$G^{(k_n,p_n)p'_n} = g_{k_n} \delta_{p_n p'_n} \tag{8}$$

where the index (k, p) represents one index obtained by vectorizing over the dimension in the parentheses, where δ is the delta function, taking value 1 only if all indexes are the same and 0 otherwise. As a result, the MPO convolution block is defined as a linear projection on a subset of the full space. Notably, using the MPO formalism it is natural to also increase the multiplicity of the kernels by increasing the MPO's rank dimension, defining the convolution MPO blocks as:

$$\mathrm{G}^{(k_n,p_n)p'_n}_{a_n a_{n+1}} = \mathrm{G}^{k_n}_{a_n a_{n+1}} \delta_{p_n p'_n}, \tag{9}$$

such that the kernels over different subspaces interact.

2.2.4. Masking MPO

The existing tensor network based polynomial regression procedures have degenerate polynomial coefficients as defined in Eq. (2) in which the weight tensor includes all orderings of multiplications of the same terms. Imposing symmetric constraint on the coefficients, is often hard to model, especially for tensor decomposition methods. Modeling the degenerate polynomial in Eq. (3) with respect to Eq. (2), increases represented coefficients of a factor of $K \simeq e^{-n} n^n$ (which for $n = 6$ is $\simeq 115$ and for $n = 10$ it is $\sim 5 \times 10^4$). For large polynomial degrees n, the divergence of K can impair the expressive power and the explainability of the model. The ideal scenario, would be to associate each combination of input (monomial) to one and only one element of the model output. We can remove the degeneracy of the coefficients by introducing a mask that filters the tensor of coefficients, obtaining the model for the polynomial defined in Eq. (2). We model the mask filtering degenerate coefficients as an MPO by defining its blocks C as

$$\mathrm{C}^{i_i i'_i}_{b_i b_i+1} = \sum_k H_{b_i k} \mathrm{D}_{k i_i i'_i b_{i+1}}, \tag{10}$$

where $H_{ij} = \theta(j-i)$, $\mathrm{D}_{ab\ldots} = \delta_{ab\ldots}$ and θ is the Heaviside function. The polynomial can consequently, be expressed as a contraction between a tensor representing coefficients, a masking MPO and a tensor containing the inputs, all retaining the block structure. Notably, due to this separate masking MPO, the gradient and subsequent Hessian calculations used for inference remain unchanged.

2.2.5. G-Ring

For the polynomial input map used in Eq. (4), the coefficients of the optimal solution are permutation invariant. We briefly provide a proof in Appendix A.4 for the regression case. In Ayvaz & Lathauwer (2021) symmetric CPD structures are explored, where symmetry (and permutation invariance) is obtained by repeating an identical block through the input space. On the same note, we define the respective counterpart for MPS/MPO formalism, the invariant ring. Due to the natural structural asymmetry of MPS structures, to obtain permutational invariance imposing all blocks identical is not enough. We need to impose also periodic boundary conditions, obtaining the invariant ring defined as:

$$\boldsymbol{R}_{l i_1 \ldots i_N} = \sum_{r_1 \ldots r_N} A^l_{r_1 i_1 r_2} \cdots A^l_{r_N i_N r_1} \in \mathbb{R}^{\times_{j=1}^N \dim(i_j)}. \tag{11}$$

Imposing such a structure lets us take advantage of the permutation invariant property of the solution to furthermore reduce the parameters of the model without compromising expressivity.

Note that in the case of regression l is equal to one. When l is greater then one, equation (11) associates to each output dimension one invariant ring.

Invariant ring structure is not trivially learnable using natural gradient, because the derivative with respect to a block is asymmetric and non-linear with respect to the block-space, causing the second derivative to be non-trivial to compute. Consequently we only use gradient descent to learn the parameters of the ring.

2.3. Efficient model inference

Canonical Polyadic Decomposition (CPD) has been a cornerstone for modeling multiway interactions due to parsimonious, often identifiable representations (Kolda & Bader, 2009). Traditionally, CPD decompositions are optimized either via ALS (block coordinate least squares) for simplicity or exploit CP structure via second–order strategies (structured Jacobian–Gramian products) (Sorber et al., 2013), damped Gauss–Newton/LM (Phan et al., 2013), and trust–region guidelines (Vervliet & Lathauwer, 2019), with inexact GN scaling to non–least–squares losses (Vandecappelle et al., 2021). In multivariate polynomial optimization, CPD has parameterized polynomial coefficients and enabled efficient Gauss-Newton (GN) implementations defining the TeMPO procedure for least squares minimization (Ayvaz & Lathauwer, 2021). MPS and MPO format provides quadratic storage in the rank, with favorable conditioning (Oseledets, 2011). For MPS/TT and MPO, traditionally training proceeds by alternatively solving subproblems defined per block of the tensor network utilizing alternating linear systems (ALS) and density matrix renormalization group (DMRG) update schemes (Holtz et al., 2012; Grasedyck et al., 2019). Using these methodologies MPS/TT models have demonstrated competitive supervised learning performance (Stoudenmire & Schwab, 2016b). Leveraging these prior works, we adopt the efficiency of natural gradient based ALS within our $(\mathrm{MPO})^2$ framework. Additionally we investigate the use of traditional gradient descent optimization, to allow efficient scaling of the number of parameters.

To our knowledge, no prior work jointly tackles *MPO-structured multivariate polynomials*, *linear, convolutional and masking operators*, *alternating natural gradient* and *gradient descent* training. Our formulation closes this gap, unifying tensorized polynomial modeling with loss agnostic optimization.

2.3.1. Alternating natural gradient

Natural gradient is a second order optimization method, that calculates the update step of the parameters taking in consideration the curvature or the loss, resulting in faster learning. Often utilized algorithms for MPOs are alternating least squares (ALS) or the density matrix renormalization group (DMRG), both sharing similar properties and methods.

Given an objective loss to minimize $\min_\theta L(y, x(\theta))$, natural gradient defines the best update of the parameters as $\Delta\theta = -\boldsymbol{H}_\theta^{-1}(L)\boldsymbol{j}_\theta(L)$, where $\boldsymbol{H}_\theta(L)$ and $\boldsymbol{j}_\theta(L)$ are the Hessian and the Jacobian of the loss with respect to the parameters. Inspired by alternating least squares methodologies on tensor networks, we learn the update step block-wise. The method reduces to computing the Hessian and Jacobian of the loss with respect to a block, updates the block according to the step, and repeats the process until all blocks are updated and then proceeds to repeat the operation in the opposite direction. We denote the full iteration as a *swipe*.

The Hessian is often singular in the first swipe due to random initialization especially when considering other losses than the least squares minimization. To stabilize the inference, we apply Tikhonov regularization (Boyd & Ong, 2009; Trefethen, 2019), with an exponentially decaying schedule for weight decay.

Importantly, for MPO structures, calculating the Hessian of a single block reduces to a trivial task. We can write the Hessian and Jacobian taking into account the regularization:

$$\boldsymbol{j}_{\mathrm{A}^{(i)}}(L) = \sum_{sl} \nabla_{\mathrm{A}^{(i)}} p_{ls}\ \partial_{p_{ls}} L(\boldsymbol{p}_s, y_s) + \lambda \mathrm{A}^{(i)}$$

$$\boldsymbol{H}_{\mathrm{A}^{(i)}}(L) = \sum_{sll'} \nabla_{\mathrm{A}^{(i)}} p_{ls}\ \nabla^T_{\mathrm{A}^{(i)}} p_{l's}\ \partial_{p_{ls}} \partial_{p_{l's}} L(\boldsymbol{p}_s, \boldsymbol{y}) + \lambda \boldsymbol{I}$$

where p_{ls} is the output of the model for sample s and output dimension l and $\boldsymbol{p}_s$ is the vector of outputs for sample s.

Block-wise learning and MPO structured coefficients simplify the Hessian since $\nabla_{\mathrm{A}^{(i)}} \nabla_{\mathrm{A}^{(i)}} p = 0$. Additionally, the gradient with respect to a block, amounts to calculating the contraction of the full MPO without the differentiated block.

Notably when using the least squares loss the natural gradient method is equivalent to the more commonly used ALS method defined in Holtz et al. (2012).

The complexity cost of the Newton method for learning is dominated by the computation of the inverse for the Hessian, which at worst, scales as the cube of the product of the dimensions of a block. To be more clear, the computational cost of the leading term C is:

$$(d)^3 N \leq C \leq (D)^3 N$$

Where d is the product of the dimensions of the smallest learnable block, D is the product of the dimensions of the largest learnable block, and N is the number of learnable blocks.

The forward pass, which is the full contraction of the network can cause the computational cost to diverge. If the contraction path is not optimized, it can lead to the multiplication of matrices with large dimensions. We assume that the contraction path are optimized. The assumption is fair since it is common for tensor networks computational frameworks to have efficient methods to find the best contraction path.

3. Related Works

Using tensor products to model general feature-output maps has a long history going back to Pi-Sigma networks (Shin & Ghosh, 1991). Recently, higher order Sigma–Pi and Sigma–Pi–Sigma neural networks (SP-SNNs) were proposed (Jiao & Su, 2024; Deng et al., 2024; Sarıkaya et al., 2023). They use multiplicative units in place of neurons and parameter sharing over layers to compactly encode polynomial maps.

Theoretically, multiplicative networks can approximate smooth targets with fewer layers/neurons than ReLU nets (Ben-Shaul et al., 2023; Jayakumar et al., 2020), while biologically inspired multiplicative couplings accelerate learning and gating in RNNs (Zhang et al., 2025b). Deep polynomial networks such as Π-nets combine hierarchical polynomial expansions with CPD (rank 1) implementations and parameter sharing across layers to curb parameter growth and achieve strong results in vision (Chrysos et al., 2022b).

Tensor Machines learn target–specific polynomial features via low–rank CPD tensors (Yang & Gittens, 2015). These typically assume CP/Tucker parameterizations and squared/logistic losses. From input output mode perspective, they can be seen as a variants of multivariate polynomial models of recent work (Ayvaz & Lathauwer, 2021).

As opposed to rank-1 models, our approach leverages the exponentially higher expressivity of MPO over CPD, as shown in Oseledets (2011). Our MPO model generalizes the aforementioned Π-nets without layer-wise non-linear activations and multivariate polynomial models by offering unifying architecture based on arbitrary rank decompositions and multilinear filters, such as convolution or (random) feature projections (Kar & Karnick, 2012).

4. Results and Discussion

We present a comparison between $(\text{MPO})^2$, CPD for polynomial regression using symmetric CPD based on TeMPO (Ayvaz & De Lathauwer, 2022) and asymmetric CPD (CPD-A) (Govindarajan et al., 2022) optimized in our framework. We further include the classical TT/MPS structure for regression both with Fourier basis (TNML-F) as in Stoudenmire & Schwab (2016b) and polynomial basis (TNML-P) as in Götte et al. (2021). The TNML-F models are optimized both using the present optimization framework and the original paper formulation, which utilizes a gradient based algorithm inspired to the density matrix renormalization group (DMRG). We additionally present the result for the model in our framework training using gradient descent with weight decay, AdamW (Loshchilov & Hutter, 2017). The model learned through the natural gradient method described in Section 2.3.1 will be indicated with N, while the one learned with gradient descent will be indicated with a G. For comparison, we also included Gaussian Processes (GP) and XGBoost (Chen & Guestrin, 2016) as implemented in scikit-learn (Pedregosa et al., 2011) as well as a multilayer perceptron (MLP) considering nine tabular datasets, five for regression and four for classification.

The datasets are chosen based on popularity in the UCML repository (Kelly et al., 2019). We perform a two level cross-validation setup with 70% training, 15% validation, and 15% test splits. Hyperparameters were tuned on the validation set, and the models were retrained with the optimal parameters before final evaluation on the test set. Dataset details as well as the full data pipeline, pre-processing as well as hyperparameter selection and optimization details for all procedures are given in Appendix A.8 together with code attached for reproducibility.

Due to the Hessian being unstable in the early phase of optimization, we applied Tikhonov regularization with an exponentially decaying schedule. To find the suitable regularization level we decay it and use early stopping to stop when validation loss does not decrease for 10 block/operator updates. In all tabular experiments we start with an initial value of $\lambda_{\text{start}} = 5$ and decay exponentially with $\gamma = 0.25$ as such: $\lambda_n = \lambda_{\text{start}} \cdot \gamma^n = 5.0 \cdot (0.25)^n$ where n is the number of trained blocks and operators so far in the optimization order.

In Table 1 we report test performance using R^2 for regression and accuracy for classification, with standard errors computed over ten random initializations when applicable. The table is organized into three groups: the proposed $(\text{MPO})^2$ models, existing tensor decomposition based polynomial models, and non-polynomial baselines for supervised learning. The best overall model is highlighted in bold, while the best polynomial tensor network based model is underlined. XGBoost was run in its default deterministic setting, and GP showed no variation across random seeds, and therefore, no error bars are reported for these two baselines. The missing results in the Table 1 for GP are due to dataset sizes where the ablation study never finished whereas the missing results for N-TNML-P are due to the model not being able to escape singularity in any seed for any model specifications we iterated over.

For TNML models, we ablate over a set hyperparameters as well as different methods and report the test result for the best performing configuration. TNML-P is learned both with our implementation of ALS, as in the original paper, and gradient descent. Additionally, for TNML-F we implemented the training method described in (Stoudenmire & Schwab, 2016b), which is a DMRG inspired gradient descent.

For both regression and classification, $(\text{MPO})^2$ outperforms the other polynomial tensor network-based methods on most datasets, and when it is not the best, its performance remains close to the strongest alternative tensor based method. Notably, feature ordering plays an important role when contrasting TNML-P and TNML-F with CPD and $(\text{MPO})^2$. Since CPD and $(\text{MPO})^2$ are invariant to feature ordering, they consistently outperform the MPS/TT structures that are feature order dependent.

Comparing with standard regression and classification models (XGBoost, MLP, GP), we see that polynomial models, probably unsurprisingly, perform worse. This is due to the expressive capabilities of the models. XGBoost and MLP can virtually represent any function due to the non-linearity intrinsic in the models, losing explainability and insight in how the data can produce the prediction. Additionally, XGBoost uses Polynomial models expressive power is constrained by the structural choice of the degree of the polynomial they are representing. However, constricting to a polynomial can retain a good enough accuracy (depend-

Table 1. Models performance comparison for regression (left datasets) and classification (right datasets) considering test quality, R^2 (%) for regression and accuracy (%) for classification (higher is better). We will indicate with G the gradient descent based learning and with N the natural gradient method, nothing if models were learned with their own code and do not fall into those categories. In the top section the model proposed in this paper, in the middle section other tensor decomposition function approximation methods, in the bottom section non polynomial models for supervised learning. **Base** refers to a baseline model that predicts the mean target value for regression tasks and the most frequent class label for classification tasks.

	Regression					Classification			
	RE	EE	CO	AB	AI	IR	HE	WQ	WI
N-(MPO)^2	78.44 ±3.07	99.77 ±0.00	84.03 ±1.57	59.87 ±0.19	41.39 ±0.30	97.83 ±3.07	60.43 ±2.86	55.65 ±0.48	99.26 ±1.56
G-(MPO)^2	81.22 ±1.14	99.58 ±0.04	84.29 ±1.56	60.16 ±0.42	40.75 ±1.89	98.70 ±2.10	59.57 ±9.06	56.09 ±1.13	**100.00** ±0.00
G-Ring	81.49 ±3.39	99.48 ±0.07	84.19 ±1.25	59.48 ±0.29	38.85 ±2.98	95.65 ±0.00	52.39 ±6.35	56.64 ±1.20	95.19 ±2.50
N-CPD-A	80.61 ±0.78	99.52 ±0.17	82.37 ±1.78	59.67 ±0.68	39.07 ±1.79	96.96 ±3.58	**60.65** ±3.15	54.37 ±1.81	99.26 ±1.56
G-CPD-A	77.90 ±11.40	99.50 ±0.12	85.05 ±1.12	59.68 ±0.56	39.03 ±1.80	98.70 ±2.10	57.83 ±4.00	56.66 ±1.29	**100.00** ±0.00
TeMPO	10.25 ±16.41	92.65 ±0.38	49.36 ±18.37	56.19 ±2.37	31.06 ±5.31	**100.00** ±0.00	58.70 ±2.17	54.13 ±1.39	**100.00** ±0.00
TNML-P	-102.92 ±291.83	99.65 ±0.13	71.46 ±7.48	-3451.49 ±4135.08	28.78 ±1.83	**100.00** ±0.00	20.43 ±11.37	28.94 ±5.33	73.70 ±16.60
TNML-F	-863.67 ±18.74	-483.35 ±0.00	-284.12 ±22.42	-279.58 ±4.57	-1.34 ±0.28	57.39 ±4.49	32.83 ±6.68	40.05 ±1.14	35.19 ±8.60
MLP	27.24 ±1.91	92.22 ±0.19	64.13 ±0.69	58.00 ±1.57	**60.58** ±2.14	97.39 ±3.89	53.48 ±1.19	60.00 ±2.01	99.26 ±1.66
XGBoost	**82.48**	**99.83**	**92.06**	55.41	59.12	**100.00**	54.35	**67.79**	**100.00**
GP	70.99	99.79	85.88	**60.40**	–	**100.00**	58.70	61.03	96.30
Base	-0.46	-0.09	-0.00	-0.08	-0.15	43.48	54.35	46.46	37.04

ing on the use case), while providing an highly detailed picture of the interaction strength of the features across different orders. This can be useful when dealing with data interpolation, where learning the parameters governing the dynamic is often the main goal as done in various applications (Zhang et al., 2025a; Bahmani et al., 2024; Tufail et al., 2021; Ogbo et al., 2026; Twarog et al., 2021; Rai et al., 2013).

Gradient methods seem to suffice to efficiently learn (MPO)^2 models. In all the experimentations of this paper, gradient methods are consistently faster and memory efficient, while obtaining comparable accuracy respect to second order methods. We report a time plot contrasting second order learning method with adamW gradient descent, reporting validation accuracy during training on MNIST dataset in Figure 5 of the Appendix.

In Table 2 provided in supplementary Appendix A.5, we systematically include an ablation study of the different modeling components of the (MPO)^2 procedure considering the Type I and Type II formulations (i.e., T1 and T2) as well as applications of the Masking (M) and Linear (L) MPOs. From the results we observe that all the specified (MPO)^2 variants produces best performance within the uncertainty bounds on at least one of the considered datasets. Consequently, the utility of the different (MPO)^2 variants are dataset dependent and the (MPO)^2 specification that is most suited for a given dataset needs to be assessed on the validation set. The choice of the best specification is highly dependent on prior knowledge of the specific problem and on the constrains we want to impose. For this reason the code and the framework allow for easy implementation of specific arbitrary MPOs structures. The MPOs presented in this paper, namely LMPO MMPO and CMPO, have been chosen to showcase how this framework can deal with uncostrained (LMPO), structured (MMPO) and hybrid (CMPO) operations on the input space. Note that MPOs structures can also represent or combine with CPDs structures. MPOs that are diagonal respect to the ranks (or that multiply

element-wise) are effectively CPD structured object.

The flexibility of the proposed $(\text{MPO})^2$ model in representing multivariate polynomials enables a systematic evaluation of tensor network architectures for multivariate polynomial regression, allowing different datasets to benefit from distinct structural constraints.

In the appendix we present additional experiments and provide a details on the learning specifications used in the paper. The following contributions are added:

- Expand on the derivation of the equations for the proposed MPOs structures.
- Intuitively present a justification for the proposed invariant ring for the regression case.
- Report the table containing the ablation experiments used to choose the best parameters for each model.
- Report the average test accuracy on the MNIST, FashionMNIST and CIFAR10 and CIFAR100 in function of number of parameters and report a comparison between TNML models, CPD, $(\text{CMPO})^2$ and CNN in Appendix A.6.
- Explore exact polynomial inference, and devise an efficient structure identification procedure systematically growing the polynomial degree from lower degree learned $(\text{MPO})^2$ representations that naturally avoids overfitting when considering the modeling of noise-free polynomial functions in Appendix A.7.
- Detail the specific of the experiments and datasets

5. Conclusions

We presented the $(\text{MPO})^2$ procedure for multivariate polynomial regression and demonstrated that this approach outperformed conventional tensor network based polynomial regression modeling procedures based on existing CPD and MPS/TT based decompositions for polynomial regression. We attribute the enhanced performance to the $(\text{MPO})^2$ procedures to feature order independence when compared to existing MPS/TT based methodologies. Notably, we explored the versatility of the $(\text{MPO})^2$ framework leveraging Linear, Convolutional and Masking MPO to learn compressed feature representations and accounting for weight redundancies. We expect there are many further generalizations in which the MPO formalism can be used to accommodate other types of operations. As such, we also expect the $(\text{MPO})^2$ can be a useful tool when combined with deep learning modeling approaches akin to how the pi-sigma based CPD procedure has been imposed as nonlinear polynomial transformations of deep learning models.

Limitations We presently only considered $(\text{MPO})^2$ modeling procedures in which the rank was specified to be identical across the MPO blocks. Future work should consider how individual ranks can be efficiently learned which would require an exponential evaluation of model specifications. It should also explore how Bayesian inference procedures can be used to quantify parameter uncertainty and automatically learn the relevance of different rank terms, see also Hinrich et al. (2020); Kilic & Batselier (2025). Pure end to end polynomial models can not outperform non-linear deep learning models in regression and classification, due to the limited representation power of the function space. As seen in polynomial models like Π-nets (Chrysos et al., 2022b), for image classification the results are highly improved with introduction of a non-linearity. We do not use non-linear transformations to represent the parameters of the polynomial while non-linear interactions in the features could greatly improve the results.

All the code to reproduce the results can be found at `https://github.com/0AnonymousSubmission/icml2026_submission/`

Impact Statement

This paper presents work whose goal is to advance the field of Machine Learning. There are many potential societal consequences of our work, none which we feel must be specifically highlighted here.

References

Ayvaz, M. and De Lathauwer, L. CPD-Structured Multivariate Polynomial Optimization. *Frontiers in Applied Mathematics and Statistics*, 8, 3 2022. doi: 10.3389/fams.2022.836433. URL https://doi.org/10.3389%2Ffams.2022.836433.

Ayvaz, M. and Lathauwer, L. D. Tensor-based multivariate polynomial optimization with application in blind identification. In *Proc. European Signal Processing Conference (EUSIPCO)*, pp. 1080–1084, 2021. doi: 10.23919/eusipco54536.2021.9616070.

Bahmani, B., Suh, H. S., and Sun, W. Discovering interpretable elastoplasticity models via the neural polynomial method enabled symbolic regressions. *Computer Methods in Applied Mechanics and Engineering*, 422:116827, March 2024. ISSN 0045-7825. doi: 10.1016/j.cma.2024.116827. URL http://dx.doi.org/10.1016/j.cma.2024.116827.

Ben-Shaul, I., Galanti, T., and Dekel, S. Exploring the approximation capabilities of multiplicative neural networks for smooth functions. *arXiv preprint arXiv:2301.04605*, 2023. URL https://arxiv.org/abs/2301.04605.

Bishop, C. M. *Pattern recognition and machine learning*, volume 4. Springer, 2006.

Boyd, J. P. and Ong, J. R. Exponentially-convergent strategies for defeating the runge phenomenon for the approximation of non-periodic functions, part i: single-interval schemes. *Comput. Phys*, 5(2-4):484–497, 2009.

Chen, T. and Guestrin, C. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pp. 785–794, 2016.

Chrysos, G. G., Georgopoulos, M., Deng, J., Kossaifi, J., Panagakis, Y., and Anandkumar, A. Augmenting deep classifiers with polynomial neural networks. In *European Conference on Computer Vision*, pp. 692–716. Springer, 2022a.

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

Dauphin, Y. N., Fan, A., Auli, M., and Grangier, D. Language modeling with gated convolutional networks. In *International conference on machine learning*, pp. 933–941. Pmlr, 2017.

Deng, F., Liang, S., Qian, K., Yu, J., and Li, X. A recurrent sigma-pi-sigma neural network. *Scientific Reports*, 14:84299, 2024. doi: 10.1038/s41598-024-84299-y.

Deng, L. The mnist database of handwritten digit images for machine learning research [best of the web]. *IEEE signal processing magazine*, 29(6):141–142, 2012.

Efthymiou, S., Hidary, J., and Leichenauer, S. Tensornetwork for machine learning, 2019. URL https://arxiv.org/abs/1906.06329.

Götte, M., Schneider, R., and Trunschke, P. A block-sparse tensor train format for sample-efficient high-dimensional polynomial regression, 2021. URL https://arxiv.org/abs/2104.14255.

Govindarajan, N., Vervliet, N., and De Lathauwer, L. Regression and classification with spline-based separable expansions. *Frontiers in big Data*, 5:688496, 2022.

Grasedyck, L., Kluge, M., and Krämer, S. Alternating least squares tensor completion in the tt-format. *SIAM Journal on Scientific Computing*, 2019. URL https://arxiv.org/abs/1509.00311. arXiv:1509.00311.

Hendrikx, S., Boussé, M., Vervliet, N., and De Lathauwer, L. Algebraic and optimization based algorithms for multivariate regression using symmetric tensor decomposition. In *2019 IEEE 8th International Workshop on Computational Advances in Multi-Sensor Adaptive Processing (CAMSAP)*, pp. 475–479. Ieee, 2019.

Hinrich, J. L., Madsen, K. H., and Mørup, M. The probabilistic tensor decomposition toolbox. *Machine Learning: Science and Technology*, 1 (2):025011, jun 2020. doi: 10.1088/2632-2153/ab8241. URL https://dx.doi.org/10.1088/2632-2153/ab8241.

Hochreiter, S. and Schmidhuber, J. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

Holtz, S., Rohwedder, T., and Schneider, R. The Alternating Linear Scheme for Tensor Optimization in the Tensor Train Format. *SIAM Journal on Scientific Computing*, 34(2):A683–a713, 1 2012. doi: 10.1137/100818893. URL https://doi.org/10.1137%2F100818893.

Hornik, K., Stinchcombe, M., and White, H. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

Jayakumar, S. M., Czarnecki, W. M., Menick, J., Schwarz, J., Rae, J., Osindero, S., Teh, Y. W., Harley, T., and Pascanu, R. Multiplicative interactions and where to find them. In *International conference on learning representations*, 2020.

Jiao, J. and Su, K. A new sigma-pi-sigma neural network based on l1 and l2 regularization and applications. *AIMS Mathematics*, 9(3):5995–6012, 2024. doi: 10.3934/math.2024293. URL https://www.aimspress.com/aimspress-data/math/2024/3/PDF/math-09-03-293.pdf.

Kar, P. and Karnick, H. Random feature maps for dot product kernels. In *Aistats*, volume 22 of *Jmlr: W&cp*, pp. 583–591, 2012. URL https://proceedings.mlr.press/v22/kar12/kar12.pdf.

Kelly, M., Longjohn, R., and Nottingham, K. The uci machine learning repository. https://archive.ics.uci.edu, 2019.

Kilic, A. and Batselier, K. Interpretable bayesian tensor network kernel machines with automatic rank and feature selection, 2025. URL https://arxiv.org/abs/2507.11136.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Kolda, T. G. and Bader, B. W. Tensor decompositions and applications. *SIAM review*, 51(3):455–500, 2009.

Li, C.-K. A sigma-pi-sigma neural network (spsnn). *Neural Processing Letters*, 17:1–19, 2003.

Lin, H. W., Tegmark, M., and Rolnick, D. Why does deep and cheap learning work so well? *Journal of Statistical Physics*, 168:1223–1247, 2017.

Liu, Y., Liu, J., Long, Z., Zhu, C., Liu, Y., Liu, J., Long, Z., and Zhu, C. *Tensor regression*. Springer, 2022.

Loshchilov, I. and Hutter, F. Fixing weight decay regularization in adam. *CoRR*, abs/1711.05101, 2017. URL http://arxiv.org/abs/1711.05101.

Ogbo, O. S., Momoh, E. O., Ndububa, E. E., Abdulkarim, B. I., Amuda, A. G., and Afolayan, O. O. Multi-objective optimisation of the mechanical properties of rice husk ash–modified lateritic concrete. *European Journal of Environmental and Civil Engineering*, 30(1):2663490, 2026. doi: 10.1080/19648189.2026.2663490. URL https://doi.org/10.1080/19648189.2026.2663490.

Olshausen, B. A. and Field, D. J. Emergence of simple-cell receptive field properties by learning a sparse code for natural images. *Nature*, 381(6583):607–609, 1996.

Oseledets, I. V. Tensor-train decomposition. *SIAM Journal on Scientific Computing*, 33(5):2295–2317, 2011. doi: 10.1137/090752286.

Panagakis, Y., Kossaifi, J., Chrysos, G. G., Oldfield, J., Nicolaou, M. A., Anandkumar, A., and Zafeiriou, S. Tensor methods in computer vision and deep learning. *Proceedings of the IEEE*, 109(5):863–890, 2021.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Phan, A.-H., Tichavský, P., and Cichocki, A. Low complexity damped gauss–newton algorithms for candecomp/parafac. *SIAM Journal on Matrix Analysis and Applications*, 34(1):126–147, 2013. doi: 10.1137/100808034.

Rai, K., Johns, R. T., Delshad, M., Lake, L. W., and Goudarzi, A. Oil-recovery predictions for surfactant polymer flooding. *Journal of Petroleum Science and Engineering*, 112:341–350, 2013. ISSN 0920-4105. doi: https://doi.org/10.1016/j.petrol.2013.11.028. URL https://www.sciencedirect.com/science/article/pii/S0920410513003446.

Sarıkaya, C., Bas, E., and Egrioglu, E. Training sigma-pi neural networks with the grey wolf optimization algorithm. *Granular Computing*, 8(5):981–989, 2023.

Shin, Y. and Ghosh, J. The pi-sigma network: An efficient higher-order neural network for pattern classification and function approximation. In *IJCNN-91-Seattle international joint conference on neural networks*, volume 1, pp. 13–18. Ieee, 1991.

Shin, Y. and Ghosh, J. Ridge polynomial networks. *IEEE Transactions on neural networks*, 6(3):610–622, 1995.

Sorber, L., Barel, M. V., and Lathauwer, L. D. Optimization-based algorithms for tensor decompositions: Cpd, decomposition in rank-$(l_r, l_r, 1)$ terms, and a new generalization. *SIAM Journal on Optimization*, 23(2):695–720, 2013. doi: 10.1137/120868323.

Stoudenmire, E. and Schwab, D. J. Supervised learning with tensor networks. *Advances in neural information processing systems*, 29, 2016a.

Stoudenmire, E. M. and Schwab, D. J. Supervised learning with quantum-inspired tensor networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 29, pp. 4799–4807, 2016b. URL https://arxiv.org/abs/1605.05775.

Tran, D., Ranganath, R., and Blei, D. M. The variational gaussian process. *Iclr*, 2016.

Trefethen, L. N. *Approximation theory and approximation practice, extended edition*. Siam, 2019.

Tufail, M. R., Jamshaid, H., Mishra, R., Hussain, U., Tichy, M., and Muller, M. Characterization of hybrid composites with polyester waste fibers, olive root fibers and coir pith micro-particles using mixture design analysis for structural applications. *Polymers*, 13(14), 2021. ISSN 2073-4360. doi: 10.3390/polym13142291. URL https://www.mdpi.com/2073-4360/13/14/2291.

Twarog, N. R., Martinez, N. E., Gartrell, J., Xie, J., Tinkle, C. L., and Shelat, A. A. Using response surface models to analyze drug combinations. *Drug Discovery Today*, 26(8):2014–2024, 2021. ISSN 1359-6446. doi: https://doi.org/10.1016/j.drudis.2021.06.002. URL https://www.sciencedirect.com/science/article/pii/S1359644621002749.

Vandecappelle, M., Vervliet, N., and Lathauwer, L. D. Inexact generalized gauss–newton for scaling the canonical polyadic decomposition with non-least-squares cost functions. *IEEE Journal of Selected Topics in Signal Processing*, 15(3):491–505, 2021. doi: 10.1109/jstsp.2020.3045911.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Vervliet, N. and Lathauwer, L. D. Numerical optimization based algorithms for data fusion. In *Data Fusion Methodology and Applications*, chapter 4, pp. 81–128. Elsevier, 2019.

Williams, C. K. and Rasmussen, C. E. *Gaussian processes for machine learning*, volume 2. MIT press Cambridge, MA, 2006.

Xiao, H., Rasul, K., and Vollgraf, R. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.

Yang, J. and Gittens, A. Tensor machines for learning target-specific polynomial features. *arXiv preprint arXiv:1504.01697*, 2015. URL https://arxiv.org/abs/1504.01697.

Zhang, Q., Chen, S., Chen, Y., Yu, S., Feng, B., and Gao, W. Optimization of cement-slag-based stabilizer proportions and macro-micro properties research of solidified soil. *Processes*, 13(12), 2025a. ISSN 2227-9717. doi: 10.3390/pr13123855. URL https://www.mdpi.com/2227-9717/13/12/3855.

Zhang, X., Altrabulsi, M., Xu, W., Wimmer, R., Halassa, M. M., and Chen, Z. S. Multiplicative couplings facilitate rapid learning and information gating in recurrent neural networks. *bioRxiv*, 2025b. doi: 10.1101/2025.07.11.663676.

A. Appendix

A.1. Operators and Convolutions

We write the same equations as in the main part providing additional mathematical steps, graphical representations and intuitions.

One of the most interesting advantage of using tensor network representation is the representation of large spaces (the space of all polynomial coefficients) into a combination of smaller, identifiable subspaces. The deconstruction allows us to operate on the subspace to easily apply globally a more complex operation. The operators are simply MPOs placed between the input and the train blocks. Given the operator O, input matrix $\boldsymbol{X}$ and vector coefficient $\boldsymbol{a}$, the model representation of the polynomial can be written as:

$$
\begin{aligned}
&p(\boldsymbol{X}) = \boldsymbol{X}\boldsymbol{O}\boldsymbol{a} = \\
&= \sum_{\boldsymbol{d}\boldsymbol{d}'}^{d} \mathrm{A}^{\boldsymbol{d}'} \mathrm{O}^{\boldsymbol{d}}_{\boldsymbol{d}'} \mathrm{X}_{\boldsymbol{d}} = \\
&= \sum_{\boldsymbol{a}\boldsymbol{a}'} \left(\sum_{d_1 d_1'} \mathrm{A}^{d_1'}_{1 a_2} \mathrm{O}^{d_1' d_1}_{1 a_2'} \tilde{x}_{d_1} \right) \cdots \left(\sum_{d_N d_N'} \mathrm{A}^{d_N'}_{a_N 1} \mathrm{O}^{d_N' d_N}_{a_N' 1} \tilde{x}_{d_N} \right) = \\
&= \sum_{\boldsymbol{a}\boldsymbol{a}'} \prod_{j}^{N} \sum_{d_j d_j'} \mathrm{A}^{d_j'}_{a_j a_{j+1}} \mathrm{O}^{d_j' d_j}_{a_j' a_{j+1}'} \tilde{x}_{d_j}
\end{aligned}
\tag{12}
$$

Globally, the operator acts as a linear transformation of the input, although, due to the decomposition, imposing a structure on the blocks of the MPO can represent many useful transformations.

We present two different operators, the linear operator and the masking operator.

A.2. Linear Projection Operator

The operator in its most general form as in (12) can be randomly initialized and learned blockwise in the same fashion as the train structured coefficients are learned, in such a way that the model learns the best transformation of the inputs.

Especially for high dimensional inputs, computing and inverting the Hessian of a block can be challenging, and, if the inputs show linear dependency, wasteful. We introduce a learnable operator that applies a simple linear transformation for each subspace represented by the blocks, which results in global structured linear transformation. To further reduce the parameters, we can impose independency between the subspaces for the linear transformation setting the rank to one. The advantage is that the new model, instead of representing the coefficients of the polynomial with blocks of dimension $R^2 D$, instead is represented by two blocks of dimensions $R^2 D'$ and DD', where D is the dimension of the input and D' is the dimension of the projected subspace.

We define the linear MPO block as a randomly initialized learnable tensor $\mathrm{B}^{i_i i_i'}_{r_i r_{i+1}}$. When the dimension of the rank r is 1 we retrieve linear independent transformations of the inputs.

Transforming the weights of the model as

$$
\mathrm{T}_{\boldsymbol{d}} = \sum_{d_1'} (\mathrm{A}^{(1)})_{d_1'} B_{d_1' d_1} \cdots \sum_{d_N'} (\mathrm{A}^{(N)})_{d_N} B_{d_N' d_N} \tag{13}
$$

A special type of linear projection is the convolution operator. We can represent our inputs as a two dimensional tensor, and project the inputs over one of the two dimensions through a convolution. If we think at images, we can define the two dimensions as patches and pixels in each patch, respectively called p and k in the following, the classical convolution would be written has

$$x_p = \sum_k g_k X_{k,p} \tag{14}$$

and the resulting polynomial can be written as:

$$\begin{aligned} p(x_1, \dots, x_P) &= \sum_{\boldsymbol{p}}^{P} \sum_{\boldsymbol{a}} x_{p_1} A^{p_1}_{1a_2} \dots x_{p_n} A^{p_n}_{a_n 1} = \\ &= \sum_{\boldsymbol{p}}^{P} \sum_{\boldsymbol{a}} \sum_{k_1} g_{k_1} X_{k_1,p_1} A^{p_1}_{1a_2} \cdots \sum_{k_n} g_{k_n} X_{k_n p_n} A^{p_n}_{a_n 1} = \end{aligned} \tag{15}$$

Note that when the convolution kernels $\boldsymbol{g}$ are different we cannot strictly speak of a polynomial respect to the patches, since the inputs will be different in each block, but it will still be a polynomial over the full pixels space

To rewrite the (15) as an MPO we can reorder the elements, add a summation over a delta function and vectorizing the inputs to obtain

$$\begin{aligned} p(x_1, \dots, x_P) &= \sum_{\boldsymbol{p}}^{P} \sum_{\boldsymbol{p}'}^{P} \sum_{\boldsymbol{a}} \sum_{k_1} X_{k_1 p_1} g_{k_1} \delta_{p_1 p_1'} A^{p_1'}_{1a_2} \cdots \sum_{k_n} X_{k_n p_n} g_{k_n} \delta_{p_n p_n'} A^{p_n'}_{a_n 1} = \\ &= \sum_{\boldsymbol{j}}^{f} \sum_{\boldsymbol{p}'}^{P} \sum_{\boldsymbol{a}} x_{(k_1,p_1)} G^{(k_1,p_1)p_1'} A^{p_1'}_{1a_2} \dots x_{(k_n,p_n)} G^{(k_n,p_n)p_n'} A^{p_n'}_{a_n 1} \end{aligned} \tag{16}$$

where the index (k, p) represent one index obtained by vectorizing over the dimension in the parentheses, and we have defined the MPO convolution block as:

$$G^{(k_n,p_n)p_n'} = g_{k_n} \delta_{p_n p_n'} \tag{17}$$

The convolution can be seen as a linear projection on a subset of the full space.

It is possible also, to increase the multiplicity of the kernels by increasing the MPO's rank dimension, defining the MPO convolution blocks as

$$G^{(k_n,p_n)p_n'}_{a_n a_{n+1}} = \mathrm{G}^{k_n}_{a_n a_{n+1}} \delta_{p_n p_n'} \tag{18}$$

where now the kernels over different subspaces interact.

A.3. Masking Operator

Notably, The number of coefficients for a multivariate polynomial scales as:

$$N_{Sym} = \binom{M+D}{M} = \frac{(M+D)!}{D!M!} \qquad N_{Asym} \approx M^D \tag{19}$$

Imposing symmetric constraint on the coefficients, is often hard to model, especially for tensor decomposition methods. For this reason in the modeling of polynomial the symmetry is disregarded, leading to a number of represented parameters that in the worst case scales as depending on the model specifications.

Using Sterling approximation for factorials we can calculate the fraction between N_{asym}, the full degenerate space, and N_{sym}, the non degenerate space, defined in (19).

$$\frac{N_{asym}}{N_{sym}} := K \simeq \left(1 + \frac{n}{d}\right)^{-(d+n+\frac{1}{2})} n^{n+\frac{1}{2}} \sqrt{2\pi} \tag{20}$$

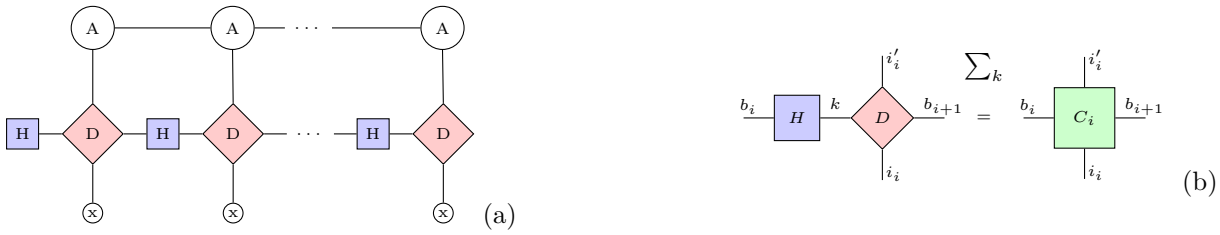


Figure 3. (a) Diagrammatic representation of the masking operator in (23). (b) Diagrammatic representation of a block of the masking MPO in (25).

for $b, n \gg 1$ (p.s. For $d > 8$ already works). Maintaining n fixed, $\lim_{d\to\infty} K \simeq e^{-n} n^n$ (which for $n = 6$ is $\simeq 115$ and for $n = 10$ it is $\sim 1/2 10^5$).

Intuitively, this points out that the train learns to represent a tensor with K times more elements than the sufficient ones. For large polynomial degrees n, the divergence of K can impair the expression power of the model and definitely hinder the explainability of the model.

The ideal scenario, would be to associate each combination of input (monomial) to one and only one element of the train output. We can achieve this by introducing a mask that filters the tensor of coefficients.

$$y = \sum_{\boldsymbol{a}}^{r} \sum_{d_1=0}^{D} A^{d_1}_{1a_2} \tilde{x}_{d_1} \sum_{d_2=d_1}^{D} A^{d_2}_{a_2a_3} \tilde{x}_{d_2} \cdots \sum_{d_n=d_{n-1}}^{D} A^{d_n}_{a_n1} \tilde{x}_{d_n} \tag{21}$$

We wish to rewrite the action in the form of a MPO. Doing so let us retain the block structure of the problem needed to use the block-wise learning algorithm. We define two auxiliary tensors, the Heaviside tensor H, and the hyperdiagonal one D.

$$\begin{aligned} H_{ij} &= \theta(j-i) \\ D_{ab\ldots} &= \delta_{ab\ldots} \end{aligned} \tag{22}$$

Where θ represents the Heaviside function and δ is a function that it is 1 only if all indexes are the same and 0 otherwise.

The equation can be rewritten:

$$\begin{aligned} p(x_1, \ldots, x_n) = \sum_{\boldsymbol{a}}^{r} \sum_{\boldsymbol{b}}^{D} \sum_{d_1,k,d_1'}^{D} A^{d_1}_{1a_2} H_{0k} D_{kd_1d_1'b_2} \tilde{x}_{d_1'} \\ \sum_{d_2,k,d_2'}^{D} A^{d_2}_{a_2a_3} H_{b_2k} D_{kd_2d_2'b_3} \tilde{x}_{d_2'} \cdots \sum_{d_n,k,d_n'}^{D} A^{d_n}_{a_n1} H_{b_nk} D_{kd_nd_n'} \tilde{x}_{d_n'} \end{aligned} \tag{23}$$

We can now extract an MPO by defining its blocks.

$$C^{d_id_i'}_{b_ib_i+1} = \sum_k H_{b_ik} D_{kd_id_i'b_{i+1}} \tag{24}$$

diagrammatically represented in Fig. 3

Finally obtaining the masking MPO.

$$C^{\boldsymbol{d}}_{\boldsymbol{d}'} = \sum_{b_2,\ldots,b_n} C^{d_1d_1'}_{1b_2} C^{d_2d_2'}_{1b_2} \ldots C^{d_nd_n'}_{b_n1} \tag{25}$$

represented in Fig. 3.

The polynomial can now be expressed as a contraction between a tensor representing coefficients, a mask given by the masking MPO and a tensor containing the inputs, all retaining the block structure.

$$p(x_1, \ldots, x_n) = \sum_{\boldsymbol{d}'}^{D} \sum_{\boldsymbol{d}}^{D} A^{\boldsymbol{d}'} C^{\boldsymbol{d}}_{\boldsymbol{d}'} X_{\boldsymbol{d}} \tag{26}$$

Notably, due to the separation between the masking MPO, the gradient and subsequent hessian calculation remains unchanged.

A.4. Symmetric solution space

To utilize the high symmetry in the input basis, we can study the induced properties of the solution space.

Consider the polynomial map defined by

$$\phi(\cdot)_N : \mathbb{R}^d \to \mathbb{R}^{\times_{i=1}^{N}(d+1)} \qquad \phi(\boldsymbol{x}) = \bigotimes_{i=1}^{N} \tilde{\boldsymbol{x}} \tag{27}$$

Where $\tilde{\boldsymbol{x}}$ is $\boldsymbol{x}$ with the value 1 appended. Note that any transformation of $\boldsymbol{x}$ would not change the following argument, the important part is the fact that $\phi(\cdot)$ is a tensor product of identical elements.

The resulting tensor $\phi(\boldsymbol{x}) = X$ is invariant under permutation.

$$P_{ij} X = X \tag{28}$$

Where P_{ij} is the standard permutation operator that switches dimension i with j (This follows from the definition of ϕ, where switching identical objects in the outer product, lead to the same result).

Definition A.1. For any field F We define an invariant ring of order N as the tensor network

$$\begin{aligned} \boldsymbol{A} &\in F^{d_r \times d_p \times d_r} \\ \boldsymbol{R}_{i_1 \ldots i_N} &= \sum_{r_1 \ldots r_N} A_{r_1 i_1 r_2} \cdots A_{r_N i_N r_1} \in F^{\times_{i=1}^{N} d_p} \end{aligned}$$

Note that when contracted with some input $\boldsymbol{x} \otimes \cdots \otimes \boldsymbol{x}$ can be written as a trace

$$\begin{aligned} r_N &= \sum_{i_1 \ldots i_N} \boldsymbol{R}_{i_1 \ldots i_N} \boldsymbol{x}_{i_1} \cdots \boldsymbol{x}_{i_N} \\ &= \sum_{r_1 \ldots r_N} \sum_{i_1 \ldots i_N} A_{r_1 i_1 r_2} \boldsymbol{x}_{i_1} \cdots A_{r_N i_N r_1} \boldsymbol{x}_{i_N} \\ &= trace\left(\left(\sum_i \boldsymbol{A}_i \boldsymbol{x}_i \right)^N \right) \end{aligned} \tag{29}$$

The theorem and the proof is in a preliminary state and uses a notation not well defined previously but that is convenient to the author. The main idea is that if the inputs are permutation invariant, the best solution must be permutation invariant too, and the ring structure fulfills this property.

The concept and the proof are simple, but they lay ground for future works. Firstly write the proof properly for any field and permutations types (cyclic). Secondly using the ring invariant structures will hopefully lead to be able to connect to the Waring's problem for polynomials.

Very summarized notation, in the real field $|\cdot\rangle$ is representing a vector $\langle\cdot|$, its transpose and $X^\dagger$ the conjugate transpose of X, which is just the transpose for real values.

Theorem A.2. *Let $|t\rangle$ be the coefficients of the polynomial, let X_s be the sample s feature map, let X_s be invariant under any permutation, and let y_s be the target value for sample s. Then a subspace of solutions minimizing the least square energy for $|t\rangle$, such that*

$$|t\rangle^* = argmin \sum_s \| \langle x_s|t\rangle - y_s\|$$

is defined by the space of invariant rings.

Proof. In this proof we will characterized the property of the induced space of solution and then show that the ring structure fulfill this space

setting the gradient respect of $|t\rangle$ of the squared loss we obtain for the $|t\rangle$

$$\frac{\partial}{\partial |t\rangle} \sum_s \| \langle x_s|t\rangle^* - y_s\|^2 = 2\sum_s |x_s\rangle \langle x_s|t\rangle^* - 2\sum_s y_s |x_s\rangle = 0$$

$$\sum_s |x_s\rangle \langle x_s|t\rangle^* = \sum_s y_s |x_s\rangle$$

Now we can study the property of $\sum_s |x_s\rangle \langle x_s| = \boldsymbol{X}$.

Remembering that $P_{ij} |x_s\rangle = |x_s\rangle \quad \forall i,j \in \{1 \dots N\}$.

$$\begin{aligned} \boldsymbol{X} P_{ij}^\dagger &= \sum_s |x_s\rangle \langle x_s| P_{ij}^\dagger \\ &= \sum_s |x_s\rangle \langle x_s| = \boldsymbol{X} \end{aligned} \tag{30}$$

The following property holds for permutation operators $P_{ij} = P_{ij}^{-1} = P_{ij}^\dagger$

We have $\forall\, i,j \in \{1 \dots N\}$

$$\begin{aligned} \boldsymbol{X} |t\rangle^* &= \boldsymbol{X} P_{ij} |t\rangle^* = |x_s\rangle y_s \\ P_{ij} |t\rangle^* &= |t\rangle^* = \boldsymbol{X}^{-1} |x_s\rangle y_s \end{aligned} \tag{31}$$

Meaning that the optimal solution of the problem $|t\rangle^*$ is an eigenstate with $+1$ eigenvalue of all the permutation operator, (i.e. invariant under all permutations).

It is easy to verify that the invariant ring structure fulfills this property. □

The invariant can not be trivially learned using Newton method due to the weights appearing non linearly.

A.5. (MPO)^2 model ablations

In Table 2 we provide model ablations for the (MPO)^2 considering the influence of Type I and Type II model specifications (T1 and T2), as well using the Masking (M) and Linear (L) MPOs.

A.6. Images results

We report the average accuracy on the MNIST (Deng, 2012) and FashionMNIST (Xiao et al., 2017) datasets in Fig. 4, as a function of the number of parameters, comparing against the CPD Type I specification as this structure was imposed for these datasets in (Ayvaz & De Lathauwer, 2022) and TNML with the Fourier basis Efthymiou et al. (2019). Notably, for this image dataset we apply the Convolution MPO in our (MPO)^2 procedure. Inspecting the Fig. 4 we observe that the (MPO)^2, can reach strong predictive performance using

Table 2. Model Ablation Study: Test results (top part for regression and bottom part for classification) for the ablation study where, after choosing the hyperparameter on the validation set, we report the average of ten runs for different $(\text{MPO})^2$ configurations. We highlight in bold the best results and underline the second best result. T1 and T2 denotes the Type I and Type II specifications, M the application of the masking MPO to account for degeneracies in the polynomial coefficients, and L for the application of the linear MPO.

	RE	EE	CO	AB	AI	IR	HE	WQ	WI
N-(MPO)^2 T2	60.32 ±0.97	99.72 ±0.03	**92.24** ±0.39	53.02 ±0.14	37.20 ±0.76	**100.00** ±0.00	55.11 ±0.99	54.54 ±1.33	**100.00** ±0.00
G-(MPO)^2 T2	65.62 ±1.01	99.57 ±0.03	91.80 ±0.30	54.05 ±0.31	40.36 ±0.60	**100.00** ±0.00	56.89 ±3.37	56.94 ±0.46	**100.00** ±0.00
N-(MPO)^2 T1	49.76 ±2.70	99.75 ±0.03	79.57 ±3.32	53.40 ±0.29	36.03 ±1.00	**100.00** ±0.00	53.33 ±2.22	56.16 ±0.35	**100.00** ±0.00
G-(MPO)^2 T1	3.62 ±16.98	99.55 ±0.02	77.37 ±5.06	53.39 ±0.32	40.40 ±0.72	**100.00** ±0.00	56.89 ±1.99	56.84 ±0.13	**100.00** ±0.00
N-(LMPO)^2 T2	49.38 ±0.15	94.17 ±0.37	80.47 ±7.47	52.23 ±0.56	32.26 ±0.05	99.09 ±2.03	58.67 ±1.22	55.06 ±1.08	97.78 ±2.03
G-(LMPO)^2 T2	63.24 ±1.41	99.03 ±0.07	91.22 ±0.58	54.45 ±0.29	**42.87** ±0.89	**100.00** ±0.00	**60.00** ±2.72	57.37 ±0.50	**100.00** ±0.00
N-(LMPO)^2 T1	50.28 ±1.38	**99.78** ±0.01	79.36 ±1.21	53.53 ±0.14	35.11 ±0.19	**100.00** ±0.00	58.67 ±2.53	57.15 ±0.42	**100.00** ±0.00
G-(LMPO)^2 T1	59.37 ±1.19	99.54 ±0.03	90.06 ±0.28	54.55 ±0.28	42.53 ±0.57	**100.00** ±0.00	**60.00** ±2.22	57.48 ±0.19	**100.00** ±0.00
N-(MMPO)^2 T2	57.35 ±0.54	99.75 ±0.00	91.97 ±0.29	53.65 ±0.07	38.90 ±0.34	**100.00** ±0.00	56.44 ±2.53	54.46 ±1.95	**100.00** ±0.00
G-(MMPO)^2 T2	61.59 ±1.22	99.60 ±0.02	92.08 ±0.17	**54.78** ±0.26	40.94 ±0.59	**100.00** ±0.00	59.11 ±1.22	57.09 ±0.49	**100.00** ±0.00
N-(MMPO)^2 T1	50.58 ±0.87	99.76 ±0.02	81.20 ±2.21	53.63 ±0.30	37.94 ±0.68	**100.00** ±0.00	54.67 ±2.53	56.94 ±0.18	**100.00** ±0.00
G-(MMPO)^2 T1	26.34 ±9.36	99.65 ±0.02	83.01 ±1.82	53.87 ±0.18	41.45 ±0.90	**100.00** ±0.00	57.78 ±1.57	57.09 ±0.28	**100.00** ±0.00
N-CPD-A T2	57.39 ±2.62	98.98 ±0.08	86.17 ±2.21	53.80 ±0.45	37.09 ±3.34	**100.00** ±0.00	54.67 ±1.99	55.08 ±0.69	**100.00** ±0.00
G-CPD-A T2	57.51 ±3.56	99.59 ±0.04	90.86 ±0.41	53.70 ±0.35	40.00 ±0.90	**100.00** ±0.00	58.22 ±2.43	57.07 ±0.47	**100.00** ±0.00
N-CPD-A T1	56.62 ±1.00	99.66 ±0.04	78.19 ±3.52	53.17 ±0.11	37.06 ±1.85	**100.00** ±0.00	53.78 ±2.90	56.29 ±0.38	**100.00** ±0.00
G-CPD-A T1	14.84 ±18.59	99.56 ±0.03	80.82 ±2.96	53.42 ±0.41	39.41 ±0.95	**100.00** ±0.00	55.11 ±0.99	**57.97** ±0.82	**100.00** ±0.00
N-TNML-P	52.83 ±9.30	99.77 ±0.01	85.65 ±1.26	40.46 ±4.84	34.96 ±1.22	**100.00** ±0.00	35.56 ±6.29	– –	62.22 ±4.06
G-TNML-P	**65.87** ±0.93	99.63 ±0.04	88.00 ±1.83	-849.44 ±192.86	31.87 ±8.68	**100.00** ±0.00	54.67 ±5.35	33.99 ±3.50	91.11 ±5.62
N-TNML-F	-776.79 ±2.35	-566.28 ±0.00	-171.56 ±3.83	-248.55 ±3.35	-3.28 ±0.01	95.45 ±0.00	41.78 ±2.90	40.21 ±0.79	53.33 ±10.99
G-TNML-F	-762.27 ±8.10	-566.28 ±0.00	-168.91 ±6.47	-241.99 ±1.45	-2.74 ±0.08	95.45 ±0.00	43.11 ±4.04	41.35 ±0.76	60.74 ±6.20
G-Ring	64.95 ±1.13	99.55 ±0.08	91.79 ±0.65	53.79 ±0.33	40.76 ±1.02	**100.00** ±0.00	56.00 ±2.43	57.31 ±0.64	**100.00** ±0.00

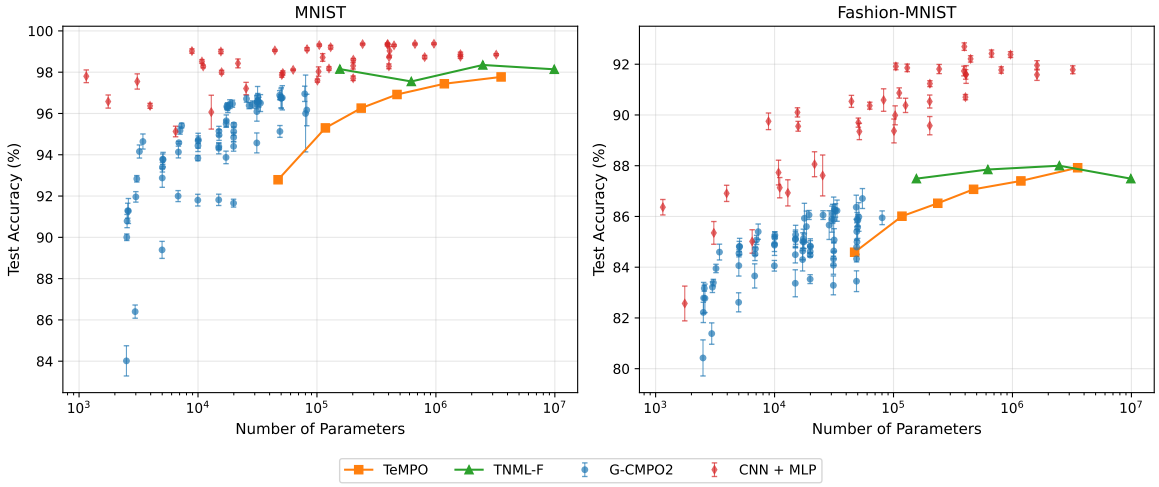


Figure 4. Accuracy on the test set for MNIST and Fashion MNIST classification tasks as function of parameters. TeMPO as described in Ayvaz & De Lathauwer (2022) as one-vs-all classifiers for each class. TNML results are reported from Efthymiou et al. (2019). CNN+MLP are different configuration of a Convolutional Network and an MLP.

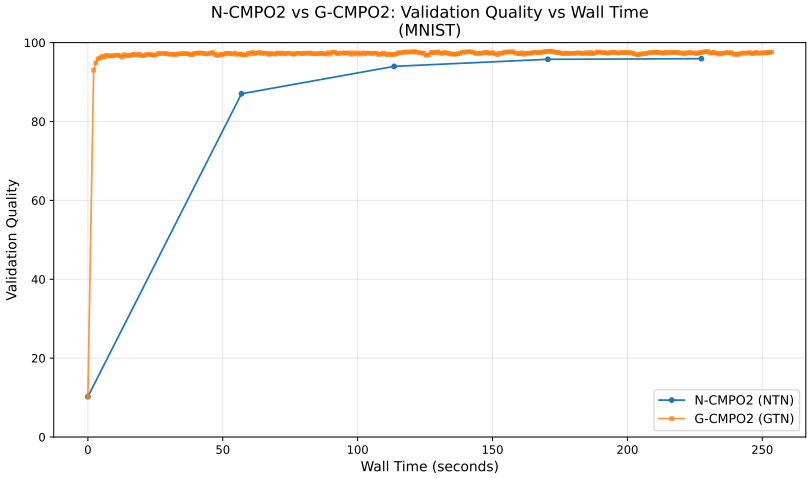


Figure 5. Validation accuracy during training on the MNIST dataset for CMPO2 model, contrasting second order learning and adamW.

substantially fewer parameters, while reaching the same results as TNML, which due to the high number of blocks being feature dependent exhibit rapidly increasing parameters as function of ranks.

For larger image datasets like CIFAR10 and CIFAR100, second order method becomes undfeasible for the high dimensionality of the inputs. Therefore we study the accuracy on the test set of the $(\text{C-MPO})^2$, learning the model parameters with gradient descent. We report in Figure 6 the accuracy in the test respect to the number of parameters of the model. The convolutional model used differ between CIFAR and MNIST datasets due to the different dimensionality of the images. CIFAR respect to MNIST has an additional channel dimension, so the patched input dimensionality is three instead of two. This means that the independent convolution tensor network structures are three, while on MNIST only two, following the same scheme for the Convolutional $(\text{MPO})^2$.

A.7. Exact polynomial inference

MPOs and polynomials suffer from computational instability, sensitivity to initialization and overfitting (Kolda & Bader, 2009; Oseledets, 2011; Ayvaz & De Lathauwer, 2022; Kilic & Batselier, 2025). We propose a growing

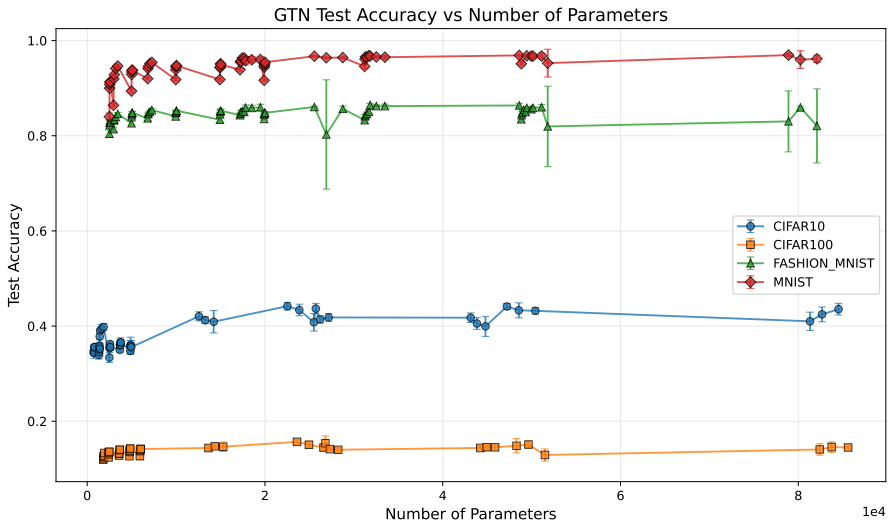


Figure 6. Test accuracy in function of number of parameters in scale 10^4 for CIFAR10, CIFAR100, MNIST and Fashion-MNIST.

learning mechanism to overcome these challenges. Long matrix multiplications are often unstable, leading to overflow or underflow, that can cause the Hessian to become singular. Additionally, when number of samples per parameter is low, models tend to overfit, cf. (Bishop, 2006).

The proposed growing method stabilizes the matrix multiplication. To implement growing, the blocks of the MPO are initialized as neutral element for the polynomial. That is, the first block is randomly initialized so that the multiplication of all the blocks to its right results in a vector of ones with rank dimension.

Formally, we will initialize the blocks:

$$\mathrm{T}^{(k)}_{i_k} = \begin{cases} \mathbf{0}, & i_k \neq 0 \\ \boldsymbol{I}, & i_k = 0 \end{cases} \quad \mathrm{T}^{(0)}_{i_0} = \begin{cases} \mathbf{0}, & i_k \neq 0 \\ \vec{1}, & i_k = 0 \end{cases} \quad \mathrm{T}^{(N)}_{i_N} = \begin{cases} \mathbf{0}, & i_k \neq 0 \\ \vec{1}, & i_k = 0 \end{cases} \tag{32}$$

Recalling that the blocks at the left and right boundaries have respectively left and right rank of dimension 1. Where $\mathbf{0}$ is a matrix full of 0 and $\vec{1}$ is a vector of ones. In practice the last block is initialized as $\mathbf{z} \sim \mathcal{N}(1, 0.02)$ instead of using $\vec{1}$. In our experiments the introduction of small noise in the initialization is essential for the learning, removing collinearity in the ranks.

The initialization in(32), introduces in the model the bias that the lowest degree polynomial fitting the data is the best. At the first update the model is trying to fit a first degree polynomial, since all blocks on the right are neutral. The effect produced is that when the block corresponding to the exact degree of the polynomial is learned, the updates of the subsequent blocks will be close to zero.

Finally, to study the effect of the growing procedure and expressivity power for exact polynomial inference, we test growing methods for $(\mathrm{MPO})^2$ and CPD against exact regression (learning the full coefficient matrix). Fig. 7 reports the average result and confidence interval over twenty randomly generated polynomial, varying number of samples, degree of the model and dimension.

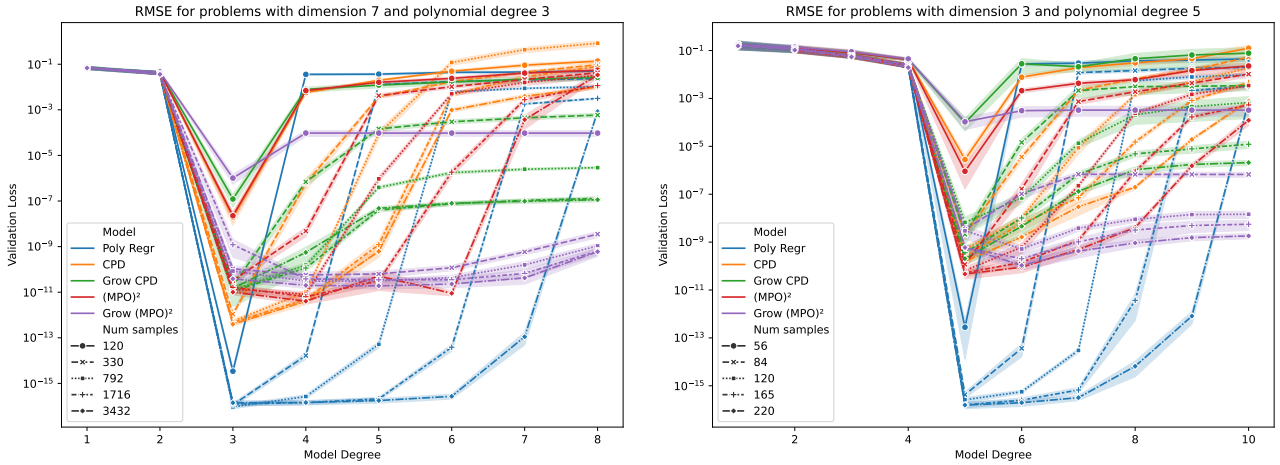


Figure 7. Validation loss across model degrees with sample sizes chosen as the binomial coefficient $\binom{N+D}{D}$, which represent the number of coefficients for a multivariate polynomial of degree N and dimension D.

A.8. Experiments Specifics

The datasets were chosen based on popularity. The historical views can be seen through these links:

- **Classification:**
  https://web.archive.org/web/20250923141238/https://archive.ics.uci.edu/datasets?Task=Classification&skip=0&take=20&sort=desc&orderBy=NumHits&search=&NumInstances=0&NumInstances=100000&NumFeatures=0&NumFeatures=100

- **Regression:**
  https://web.archive.org/web/20250923141141/https://archive.ics.uci.edu/datasets?Task=Regression&skip=0&take=20&sort=desc&orderBy=NumHits&search=&NumInstances=0&NumInstances=100000&NumFeatures=0&NumFeatures=100

The data and pre-processing pipeline is as follows for all datasets, and all datasets were processed using the same method:

Each dataset is obtained from the UCI Machine Learning Repository (Kelly et al., 2019). Feature columns containing missing values are removed. Targets are treated as a single column and converted to integer labels for classification tasks.

Feature encoding follows a capped one hot scheme to control dimensionality. Numeric columns are always kept. Categorical columns are one hot encoded, subject to a fixed maximum number of total feature columns. Columns with the largest cardinality are dropped first if the budget is exceeded. If the number of one hot encoded columns still exceeds the budget, excess variables are trimmed.

Data is split into training, validation, and test sets with proportions of 70%, 15%, and 15%, respectively. Standard normalization is used by finding the mean and standard deviation on the training set's numeric columns and applied to the corresponding validation and test columns, while one hot features remain unchanged.

Some datasets were discarded based on not fitting into this general data pipeline. We discarded the dataset diabetes[1] due to unavailability for download through the official package. We discarded Automobile[2] and Auto MPG[3] due to the small number of instances together with the presence of missing data.

Details of the number of samples and features are provided in table 3.

[1] https://archive.ics.uci.edu/dataset/34/diabetes
[2] https://archive.ics.uci.edu/dataset/10/automobile
[3] https://archive.ics.uci.edu/dataset/9/auto+mpg

Table 3. Datasets with tasks, sizes, features, and shorthand codes.

ID	Code	Dataset	Task	Train	Val	Test	Feat.
2	AD	Adult	Classification	34189	7326	7327	47
222	BA	Bank Marketing	Classification	31647	6782	6782	33
73	MU	Mushrooms	Classification	5686	1219	1219	50
186	WQ	Wine Quality	Classification	4547	975	975	11
697	SD	Students' Dropout	Classification	3096	664	664	36
19	CE	Car Evaluation	Classification	1209	259	260	27
17	BR	Breast Cancer Wisconsin	Classification	398	85	86	30
45	HE	Heart Disease	Classification	212	45	46	11
109	WI	Wine	Classification	124	27	27	13
53	IR	Iris	Classification	105	22	23	4
332	PO	Online News Popularity	Regression	27750	5947	5947	58
374	AP	Appliances Energy Prediction	Regression	13814	2960	2961	27
275	BK	Bike Sharing	Regression	12165	2607	2607	12
601	AI	AI4I	Regression	7000	1500	1500	9
560	SB	Seoul Bike Sharing	Regression	6132	1314	1314	18
1	AB	Abalone	Regression	2923	627	627	11
544	OB	Obesity Levels	Regression	1477	317	317	39
165	CO	Concrete Compressive Strength	Regression	721	154	155	8
242	EE	Energy Efficiency	Regression	537	115	116	8
320	SP	Student Performance	Regression	454	97	98	50
477	RE	Real Estate Valuation	Regression	289	62	63	6

All results from the grid search and test are provided as CSV files in the supplementary material.

A.9. Hyperparameter search

We conduct a hyperparameter grid search on the validation set for all models reported.

In this section we describe the range and model types for these searches.

A.9.1. Gaussian Process (GP)

We evaluated 14 Gaussian process kernel configurations for datasets with fewer than 4000 samples. The base kernels included a radial basis function (RBF) kernel, a Matérn kernel with smoothness parameter $\nu = 2.5$, a linear kernel, and an additive RBF–linear combination. We also tested Automatic Relevance Determination (ARD) variants of the RBF and Matérn kernels, which allow feature-specific length scales, as well as an ARD RBF combined with a linear kernel. For each of these kernels, we additionally considered versions that included a white noise term, resulting in 14 total configurations.

For larger datasets with at least 4000 samples, we restricted the search to the RBF-ARD plus linear kernel with an added white noise term due to computational limitations.

A.9.2. Multilayer Perceptron (MLP)

We conducted a grid search to optimize multi-layer perceptron (MLP) neural networks, evaluating different network architectures. The MLPs used a consistent building block of a linear transformation followed by layer normalization, a ReLU activation, and another linear transformation, repeated across the hidden layers.

The search varied the number of hidden layers (1, 3, or 5) and the number of neurons per layer (16, 64, or 256), resulting in nine distinct architectures. Each hidden layer had the same width within a given configuration. The input dimension matched the dataset features, and the output dimension was one neuron for regression tasks or the number of classes for classification tasks.

Training parameters were fixed across all runs: batch size of 256, learning rate of 0.001 with the Adam optimizer (Kingma & Ba, 2014), a maximum of 1000 epochs, and early stopping. The early stopping criterion was adaptive, with a patience of either 10 epochs or the number of input features plus one, whichever was larger.