

(MPO)²: MULTIVARIATE POLYNOMIAL OPTIMIZATION BASED ON MATRIX PRODUCT OPERATORS

Anonymous authors

Paper under double-blind review

ABSTRACT

Central to machine learning is the ability to perform universal function approximation and learn complex input-output relationships from limited numbers of observations. Suitable Multivariate Polynomial Optimization can in theory provide universal function approximations. However, the coefficients of the polynomial regression model grows exponentially in the polynomial degree. To reduce exponential growth tensor factorizations of the associated weight tensor have been explored, including the canonical polyadic decomposition (CPD) and tensor train (TT) decompositions. Whereas CPD has expressive power proportional to rank and current TT formulations are feature order dependent with each input feature associated to a specific factorization block. Furthermore, these procedures account for redundancies sub-optimally in the weight tensor. We presently explore multivariate polynomial optimization of matrix product operator (MPO) structures forming the (MPO)². Notably, the (MPO)² defines a flexible framework that naturally combines MPO polynomial weight tensors with MPO feature embeddings. The (MPO)² consequently produces an expressive yet compact representation of multivariate polynomials that is feature order independent and explicitly accounts for symmetries in the weight tensor. On a series of regression and classification problems we observe that the proposed (MPO)² provides superior performance when compared to existing tensor decomposition based multivariate polynomial regression approaches even outperforming conventional universal function approximation procedures on some datasets. The (MPO)² provides an expressive and versatile alternative to deep learning for universal function approximation with simple and efficient inference using second order methods.

1 INTRODUCTION

A central task in machine learning is to learn from data suitable functions that can map inputs to associated outputs in a way that best possible generalizes. 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, i.e. as also guaranteed by Taylor’s theorem.

In the recent decade context aware learning methods have demonstrated superior performance in generalization enabling to leverage nonlinear 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 exploring tensor decomposition structures including the canonical polyadic decomposition (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, 2016b; 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 relies 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 (MPO)² framework, a new tensor network based structure for the modeling of higher order polynomials. Notably, (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 (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.

In this work, the position of the indices of a tensor, when referring to tensor networks structures, will be at the apex to indicate vertical modes in the diagrammatic representation and at the pedex 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 \mathbf{M} with elements $M_{i_1 i_2 \dots i_n}$ will be denoted by $\sum_{i_1 i_2 \dots i_n} M_{i_1 i_2 \dots i_n} = \sum_{\mathbf{i}} M_{\mathbf{i}^{(n)}}$, where $\mathbf{i}^{(n)} = \{i_1, i_2, \dots, i_n\}$, meaning that the sum is performed over all the indices going from i_1 to i_n . When the apex is omitted, it means that $\mathbf{i} = \mathbf{i}^{(N)}$, where N is the degree of the polynomial.

A trivial example of a graphical representation of tensor networks can be seen in Figure 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.

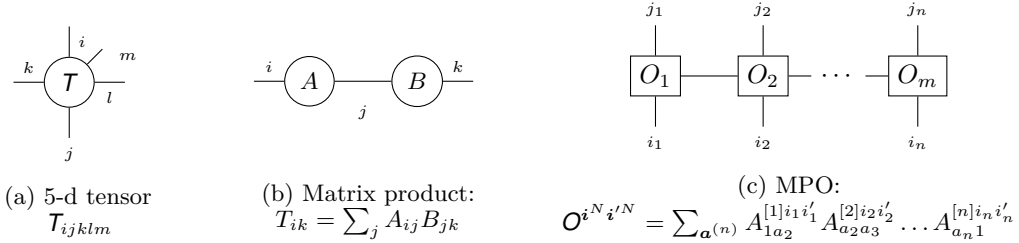


Figure 1: Graphical representation of (a) tensors, (b) the matrix product, and (c) the matrix product operator (MPO).

The well-known tensor network structures, namely *matrix product states* (MPS) and *tensor trains* (TT), as well as Tucker and CPD decompositions, are respectively given by

$$\text{MPS/TT: } T_{\mathbf{d}^{(N)}l} = \sum_{\mathbf{r}} T_{1r_2}^{[1]d_1} T_{r_2 r_3}^{[2]d_2} \dots T_{r_n 1}^{[n]d_n l} \quad (1)$$

$$\text{Tucker/CPD: } T_{\mathbf{d}^{(N)}l} = \sum_{\mathbf{r}} \mathcal{G}_{\mathbf{r}} T_{r_1}^{[1]d_1} \dots T_{r_n}^{[n]d_n}, \quad (2)$$

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 \mathbf{x} of dimension D , we define a multivariate polynomial of \mathbf{x} of degree N :

$$p_l(\mathbf{x}) = T_l + \sum_{d_1} T_{ld_1} x_{d_1} + \sum_{d_2 \geq d_1} T_{ld^{(2)}} x_{d_1} x_{d_2} + \dots + \sum_{d_2 \geq d_1} \dots \sum_{d_N \geq d_{N-1}} T_{ld^{(N)}} x_{d_1} \dots x_{d_N} \quad (3)$$

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

Following Ayzav & 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):

$$\text{Type I: } p_l(\mathbf{x}) = \sum_{n=0}^N \sum_{\mathbf{d}^{(n)}} T_{ld^{(n)}} x_{d_1} \dots x_{d_n}, \quad \text{Type II: } p_l(\tilde{\mathbf{x}}) = \sum_{\mathbf{d}^{(N)}} \tilde{T}_{ld^{(N)}} \tilde{x}_{d_1} \dots \tilde{x}_{d_N} \quad (4)$$

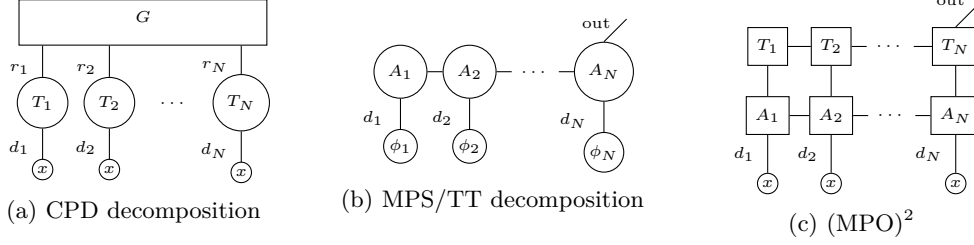


Figure 2: Existing tensor network modeling procedures for multivariate polynomial regression based on (a) the CPD decomposition, (b) the MPS/TT decomposition in which each feature has its own train-cart operating on a fixed transformation $\phi(x_i) = \phi_i$ imposed on each feature, and (c) the proposed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

where $\tilde{\mathbf{x}}$ will be the input vector \mathbf{x} with a constant additional feature of value 1 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 $\mathcal{O}(\mathcal{M}^N)$ 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 considering 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, 2016b; 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., $\mathcal{T}_{r_{m-1}r_m}^{[m]d'_m-1}$ 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 $\mathbf{A}^{[i]}$ to the inputs, we can express the polynomial as:

$$p_l(\mathbf{x}) = \sum_{\mathbf{d}\mathbf{d}'} \mathcal{T}_{\mathbf{d}'l} A_{\mathbf{d}'\mathbf{d}_1}^{[1]} x_{d_1} \dots A_{\mathbf{d}'\mathbf{d}_N}^{[N]} x_{d_N} = \sum_{\mathbf{d}\mathbf{d}'} \mathcal{T}_{\mathbf{d}'l} \mathbf{A}_{\mathbf{d}'\mathbf{d}} x_{d_1} \dots x_{d_N} \quad (5)$$

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

In the (MPO)² framework we propose to perform multivariate 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 Figure 2c and given as follows:

$$\mathcal{T}_{\mathbf{d}'^{(N)}l} = \sum_{\mathbf{r}} \mathcal{T}_{1r_2}^{[1]d'_1} \mathcal{T}_{r_2r_3}^{[2]d'_2} \dots \mathcal{T}_{r_n1}^{[n]d'_n} l, \quad \mathbf{A}_{\mathbf{d}'\mathbf{d}} = \sum_{\mathbf{r}} A_{1r_2}^{[1]d_1d'_1} A_{r_2r_3}^{[2]d_2d'_2} \dots A_{r_n1}^{[n]d_nd'_n} \quad (6)$$

where R is the rank of the MPO structure.

We propose three structures of the MPOs representing the input transformation tensor \mathbf{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 (see section 2.3) by a factor $\sim (\frac{D}{D'})^3$. The operator in its most general form as in equation 8 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 $R^2 D$, 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 $\mathbf{B}_{r_i r_{i+1}}^{i_i i'_i}$. 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. By representing the inputs as a two dimensional tensor, and project 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 (Olshausen & Field, 1996), respectively called p and k in the following. The convolution with kernel g can then be written has $x_p = \sum_k g_k X_{k,p}$. Consequently, we define the convolution MPO block by $G^{(k_n, p_n) p'_n} = g_{k_n} \delta_{p_n p'_n}$ 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 MPO convolution blocks as $\mathbf{G}_{a_n a_{n+1}}^{(k_n, p_n) p'_n} = \mathbf{G}_{a_n a_{n+1}}^{k_n} \delta_{p_n p'_n}$, 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 equation 3 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. 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. Modeling the degenerate polynomial in equation 4 respect to equation 3, increases the coefficients to be modeled 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 expression 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 achieve this by introducing a mask that filters the tensor of coefficients obtaining the model for the polynomial defined in equation 3. We model the mask filtering degenerate coefficients as an MPO by defining its blocks \mathbf{C} as $\mathbf{C}_{b_i b_{i+1}}^{i_i i'_i} = \sum_k H_{b_i k} \mathbf{D}_{k i_i i'_i b_{i+1}}$, where $H_{ij} = \theta(j - i)$, $\mathbf{D}_{ab...} = \delta_{ab...}$ and θ is the Heaviside function. The polynomial can consequently, be expressed as a contraction between a tensor representing coefficients and 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 (see section 2.3) remain unchanged.

2.3 EFFICIENT MODEL INFERENCE

In multivariate polynomial optimization, CPD has parameterized polynomial coefficients and enabled efficient Gaus-Newton (GN) implementations defining the TeMPO procedure for least squares minimization (Ayvaz & Lathauwer, 2021). 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, 2016a). Leveraging these prior works, we adopt the efficiency of natural gradient based ALS within our (MPO)² framework. To our knowledge, no prior work jointly tackles *MPO structured multivariate polynomials, linear, convolutional and masking operators*, and *alternating natural gradient* training. Our formulation closes this gap, unifying tensorized polynomial modeling with loss agnostic optimization.

2.3.1 ALTERNATING NATURAL GRADIENT

To learn the block of the MPOs we use a block-wise version of the natural gradient method, optimal for large family of Bregman divergence losses, incl., neg. log likelihood, cross entropy (classification) and least squares (regression), see Amari (2016).

Notably, standard gradient updates when dealing with MPS/TT have been demonstrated to lead to slow convergence (Qiu et al., 2024). The 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 to faster learning. Often utilized algorithms for MPOs are alternating least squares or the density matrix renormalization group, 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 $\Delta\theta = -\mathbf{H}_{\theta}^{-1}(L)\mathbf{j}_{\theta}(L)$, where $\mathbf{H}_{\theta}(L)$ and $\mathbf{j}_{\theta}(L)$ are the Hessian and the Jacobian of the loss 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 swipes, especially when considering other losses than least squares minimization, due to random initialization. To stabilize the inference, we introduce 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:

$$\begin{aligned} \mathbf{j}_{\mathbf{A}^{(i)}}(L) &= \sum_s \sum_l \nabla_{\mathbf{A}^{(i)}} p_{ls} \partial_{p_{ls}} L(\mathbf{p}_s, \mathbf{y}_s) + \lambda \mathbf{A}^{(i)} \\ \mathbf{H}_{\mathbf{A}^{(i)}}(L) &= \sum_s \sum_{l,l'} \nabla_{\mathbf{A}^{(i)}} p_{ls} \nabla_{\mathbf{A}^{(i)}}^T p_{l's} \partial_{p_{ls}} \partial_{p_{l's}} L(\mathbf{p}_s, \mathbf{y}) + \lambda \mathbf{I} \end{aligned} \quad (7)$$

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

Block-wise learning and MPO structured coefficients simplify the Hessian since $\nabla_{\mathbf{A}^{(i)}} \nabla_{\mathbf{A}^{(i)}} p = 0$. 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).

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 (SPSNNs) 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., 2025). 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), while avoiding the quadratic growth in block parameters ($\approx R^2$ for MPO versus $\approx R$ for CP). Our MPO model generalizes aforementioned Π -nets without layer nonlinear 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 (MPO)², CPD for polynomial regression using symmetric CPD (CPD-S) 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 Efthymiou et al. (2019) and polynomial basis (TNML-P) as in Götte et al. (2021). The TNML-F models are optimized using the present optimization framework to directly assess the impact on model structure on performance whereas the original paper utilizes density matrix renormalization group (DMRG). For comparison, we also included Gaussian Processes (GP) and XGBoost as implemented in scikit-learn (Pedregosa et al., 2011) as well as a multilayer perceptron (MLP) considering twenty tabular datasets, ten for regression and ten 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.6 together with code attached for reproducibility. In the hyperparameter search for CPD, we considered both the same ranks used for (MPO)² and their squared values in order to obtain models with comparable parameter counts. During the hyperparameter search, we explored a broader range of CPD ranks, including both those used for (MPO)² and their squared values to match parameter counts.

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 and Table 2 we report the average R^2 for regression and accuracy for classification, evaluated on the test set with the standard error of the mean over five random initializations of each model. In the upper part of each table we present the polynomial tensor network

based models, while the lower part contains generic function approximation models. The best overall model is highlighted in bold, and the best among the polynomial tensor network based methods is underlined. XGBoost was run in its default deterministic setting, and GP showed no variation in predictive performance across random seeds, so no error bars are reported for these two methods.

For both regression and classification, (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. Overall, (MPO)² is on average the best performing tensor network based approach. Notably, feature ordering plays an important role when contrasting TNML-P and TNML-F with CPD and (MPO)². Since CPD and (MPO)² are invariant to feature ordering, they consistently outperform the MPS/TT structures that are feature order dependent.

Table 1: Models performance comparison for regression considering R^2 (higher is better).

	SP	AB	OB	BK	RE	EE	CO	AI	PO	SB	Avg
(MPO) ²	<u>21.69</u> ±0.12	58.97 ±0.35	<u>73.53</u> ±0.55	<u>74.23</u> ±0.13	80.82 ±0.71	<u>99.78</u> ±0.01	<u>85.40</u> ±0.44	<u>42.45</u> ±0.37	<u>1.96</u> ±0.24	60.84 ±2.48	<u>59.97</u>
CPD-A	20.93 ±0.28	<u>59.82</u> ±0.07	71.18 ±2.04	73.65 ±0.21	<u>81.26</u> ±0.84	99.69 ±0.01	80.89 ±0.79	40.44 ±0.56	1.71 ±0.03	<u>68.72</u> ±2.92	59.83
CPD-S	19.60 ±2.37	56.19 ±0.75	51.86 ±0.49	41.45 ±0.12	10.25 ±5.19	92.65 ±0.12	49.36 ±5.81	31.06 ±1.68	1.13 ±0.17	44.26 ±5.38	39.78
TNML-P	-700.45 ±190.59	55.69 ±0.68	48.80 ±2.51	61.61 ±0.26	78.10 ±0.22	99.64 ±0.02	73.27 ±1.69	27.91 ±0.03	-9.06 ±0.07	20.31 ±4.49	-24.42
TNML-F	-1446.58 ±14.24	-28.05 ±1.29	-87.67 ±2.12	-20.98 ±5.62	-522.90 ±0.78	-224.25 ±0.80	-72.28 ±3.42	2.28 ±0.12	-9.06 ±0.03	-1676.04 ±2.84	-408.55
GP	21.97	60.40	94.22	–	70.99	99.79	85.88	–	–	–	72.21
MLP	18.42 ±0.33	58.00 ±0.70	89.33 ±1.20	94.25 ±0.20	27.24 ±0.85	92.22 ±0.09	64.13 ±0.31	60.58 ±0.96	2.34 ±0.01	92.24 ±0.58	59.88
XGBoost	19.61	55.41	92.12	94.84	82.48	99.83	92.06	59.12	0.72	97.81	69.40

Table 2: Performance comparison of classification models (accuracy, higher is better).

	IR	HE	WQ	BR	AD	BA	WI	CE	SD	MU	Avg
(MPO) ²	<u>100.00</u> ±0.00	<u>61.74</u> ±1.11	<u>55.30</u> ±0.65	<u>99.77</u> ±0.23	56.87 ±0.02	90.57 ±0.02	<u>100.00</u> ±0.00	<u>99.23</u> ±0.21	77.71 ±0.34	99.47 ±0.02	<u>84.07</u>
CPD-A	<u>100.00</u> ±0.00	59.13 ±0.81	55.02 ±0.25	99.07 ±0.23	<u>56.94</u> ±0.02	90.51 ±0.03	96.30 ±0.00	97.15 ±0.09	<u>78.07</u> ±0.22	99.46 ±0.03	83.16
CPD-S	<u>100.00</u> ±0.00	58.70 ±0.69	54.13 ±0.44	96.51 ±0.97	56.57 ±0.10	90.16 ±0.09	<u>100.00</u> ±0.00	82.15 ±0.99	76.87 ±0.27	99.13 ±0.03	81.42
TNML-P	<u>100.00</u> ±0.00	56.09 ±1.06	48.62 ±2.90	79.07 ±0.37	56.19 ±0.12	89.67 ±0.15	82.96 ±4.32	99.23 ±0.27	51.20 ±1.52	99.26 ±0.25	76.23
TNML-F	77.39 ±1.63	31.30 ±1.63	45.29 ±0.42	73.72 ±2.22	47.20 ±0.44	78.58 ±0.48	51.85 ±6.09	95.69 ±0.51	33.43 ±3.34	98.98 ±0.25	63.35
GP	100.00 –	58.70 –	61.03 –	97.67 –	– –	– –	96.30 –	96.54 –	78.31 –	– –	84.08
MLP	97.39 ±1.74	53.48 ±0.53	60.00 ±0.90	99.30 ±0.47	57.10 ±0.06	90.96 ±0.03	99.26 ±0.74	98.46 ±0.34	76.51 ±0.23	99.46 ±0.02	83.19
XGBoost	100.00	54.35	67.79	96.51	57.81	90.95	100.00	96.54	78.01	99.51	84.15

In Table 3 provided in supplementary section A.4, 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 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 accessed on the validation set. As such the versatile specification of the multivariate polynomial by the considered (MPO)² specifications

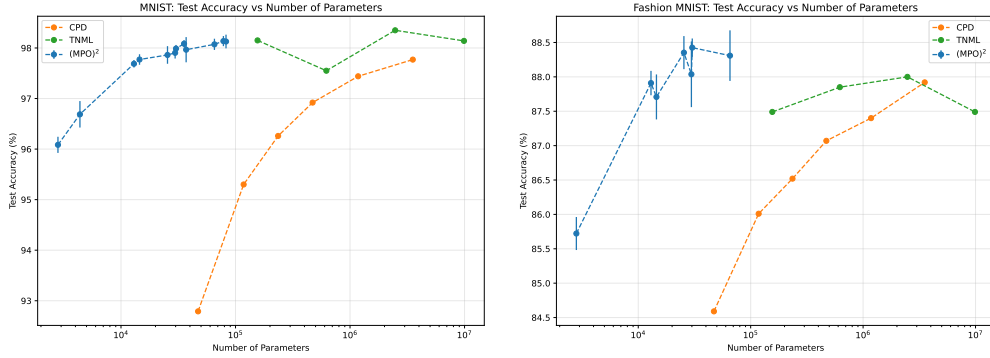


Figure 3: Accuracy on the test set for MNIST and Fashion MNIST classification tasks as function of parameters. CPD is optimized using the TeMPO procedure in Ayvaz & De Lathauwer (2022) as one-vs-all classifiers for each class. TNML results are reported from Efthymiou et al. (2019).

enable the systematic assessment of suitable tensor network specifications for multivariate polynomial regression with each dataset benefiting from different structures imposed.

We additionally report the average accuracy on the MNIST (Deng, 2012) and FashionMNIST (Xiao et al., 2017) datasets in Figure 3, 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 Figure 3 we observe that the (MPO)², can reach strong predictive performance using 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.

In the appendix, we furthermore explore exact polynomial inference, and devise an efficient inference procedure systematically growing the polynomial degree from lower degree learned (MPO)² representations that naturally avoids overfitting when considering the modeling of noise-free polynomial functions in A.5.

5 CONCLUSIONS

We presented the (MPO)² procedure for multivariate polynomial regression and demonstrated that this approach systematically 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 (MPO)² procedures higher modeling capacity when compared to CPD structures and feature order independence when comparing to existing MPS/TT based methodologies. Notably, we explored the versatility of the (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 (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 Chrysos et al. (2022b); Panagakis et al. (2021); Chrysos et al. (2022a).

Limitations We presently only considered (MPO)² modeling procedures in which the rank R 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).

REFERENCES

- Shun-ichi Amari. *Information Geometry and Its Applications*. Springer, 2016.
- Muzaffer Ayvaz and Lieven De Lathauwer. 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/fams.2022.836433>.
- Muzaffer Ayvaz and Lieven De Lathauwer. 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.
- Ido Ben-Shaul, Tomer Galanti, and Shai Dekel. Exploring the approximation capabilities of multiplicative neural networks for smooth functions. *arXiv preprint arXiv:2301.04605*, 2023. URL <https://arxiv.org/abs/2301.04605>.
- Christopher M Bishop. *Pattern recognition and machine learning*, volume 4. Springer, 2006.
- John P Boyd and Jun Rong Ong. 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.
- Grigorios G Chrysos, Markos Georgopoulos, Jiankang Deng, Jean Kossaifi, Yannis Panagakis, and Anima Anandkumar. Augmenting deep classifiers with polynomial neural networks. In *European Conference on Computer Vision*, pp. 692–716. Springer, 2022a.
- Grigorios G. Chrysos, Stylianos Moschoglou, Giorgos Bouritsas, Jiankang Deng, Yannis Panagakis, and Stefanos Zafeiriou. Deep polynomial neural networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(8):4021–4034, 2022b. doi: 10.1109/TPAMI.2021.3058891.
- Yann N Dauphin, Angela Fan, Michael Auli, and David Grangier. Language modeling with gated convolutional networks. In *International conference on machine learning*, pp. 933–941. PMLR, 2017.
- Fei Deng, Shibin Liang, Kaiguo Qian, Jing Yu, and Xuanxuan Li. A recurrent sigma-pi-sigma neural network. *Scientific Reports*, 14:84299, 2024. doi: 10.1038/s41598-024-84299-y.
- Li Deng. The mnist database of handwritten digit images for machine learning research [best of the web]. *IEEE signal processing magazine*, 29(6):141–142, 2012.
- Stavros Efthymiou, Jack Hidary, and Stefan Leichenauer. Tensornetwork for machine learning, 2019. URL <https://arxiv.org/abs/1906.06329>.
- Nithin Govindarajan, Nico Vervliet, and Lieven De Lathauwer. Regression and classification with spline-based separable expansions. *Frontiers in big Data*, 5:688496, 2022.
- Lars Grasedyck, Melanie Kluge, and Sebastian Krämer. 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.
- Michael Götte, Reinhold Schneider, and Philipp Trunschke. A block-sparse tensor train format for sample-efficient high-dimensional polynomial regression, 2021. URL <https://arxiv.org/abs/2104.14255>.
- Stijn Hendrikx, Martijn Boussé, Nico Vervliet, and Lieven De Lathauwer. 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.
- Jesper L Hinrich, Kristoffer H Madsen, and Morten Mørup. 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>.

- Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.
- Sebastian Holtz, Thorsten Rohwedder, and Reinhold Schneider. 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/10.1137/100818893>.
- Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.
- Siddhant M Jayakumar, Wojciech M Czarnecki, Jacob Menick, Jonathan Schwarz, Jack Rae, Simon Osindero, Yee Whye Teh, Tim Harley, and Razvan Pascanu. Multiplicative interactions and where to find them. In *International conference on learning representations*, 2020.
- Jianwei Jiao and Keqin Su. 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>.
- Purushottam Kar and Harish Karnick. 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>.
- Markelle Kelly, Rachel Longjohn, and Kolby Nottingham. The uci machine learning repository. <https://archive.ics.uci.edu>, 2019.
- Afra Kilic and Kim Batselier. Interpretable bayesian tensor network kernel machines with automatic rank and feature selection, 2025. URL <https://arxiv.org/abs/2507.11136>.
- Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.
- Tamara G Kolda and Brett W Bader. Tensor decompositions and applications. *SIAM review*, 51(3):455–500, 2009.
- Chien-Kuo Li. A sigma-pi-sigma neural network (spsnn). *Neural Processing Letters*, 17:1–19, 2003.
- Henry W Lin, Max Tegmark, and David Rolnick. Why does deep and cheap learning work so well? *Journal of Statistical Physics*, 168:1223–1247, 2017.
- Yipeng Liu, Jiani Liu, Zhen Long, Ce Zhu, Yipeng Liu, Jiani Liu, Zhen Long, and Ce Zhu. *Tensor regression*. Springer, 2022.
- Bruno A Olshausen and David J Field. Emergence of simple-cell receptive field properties by learning a sparse code for natural images. *Nature*, 381(6583):607–609, 1996.
- Ivan V. Oseledets. Tensor-train decomposition. *SIAM Journal on Scientific Computing*, 33(5):2295–2317, 2011. doi: 10.1137/090752286.
- Yannis Panagakis, Jean Kossaifi, Grigorios G Chrysos, James Oldfield, Mihalis A Nicolaou, Anima Anandkumar, and Stefanos Zafeiriou. Tensor methods in computer vision and deep learning. *Proceedings of the IEEE*, 109(5):863–890, 2021.
- F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.
- Shikai Qiu, Andres Potapczynski, Marc Finzi, Micah Goldblum, and Andrew Gordon Wilson. Compute better spent: Replacing dense layers with structured matrices, 2024. URL <https://arxiv.org/abs/2406.06248>.

- Cansu Sarıkaya, Eren Bas, and Erol Egrioglu. Training sigma-pi neural networks with the grey wolf optimization algorithm. *Granular Computing*, 8(5):981–989, 2023.
- Yoan Shin and Joydeep Ghosh. 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.
- Yoan Shin and Joydeep Ghosh. Ridge polynomial networks. *IEEE Transactions on neural networks*, 6(3):610–622, 1995.
- E. Miles Stoudenmire and David J. Schwab. Supervised learning with quantum-inspired tensor networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 29, pp. 4799–4807, 2016a. URL <https://arxiv.org/abs/1605.05775>.
- Edwin Stoudenmire and David J Schwab. Supervised learning with tensor networks. *Advances in neural information processing systems*, 29, 2016b.
- Dustin Tran, Rajesh Ranganath, and David M Blei. The variational gaussian process. *ICLR*, 2016.
- Lloyd N Trefethen. *Approximation theory and approximation practice, extended edition*. SIAM, 2019.
- Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.
- Christopher KI Williams and Carl Edward Rasmussen. *Gaussian processes for machine learning*, volume 2. MIT press Cambridge, MA, 2006.
- Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.
- Jiyan Yang and Alex Gittens. Tensor machines for learning target-specific polynomial features. *arXiv preprint arXiv:1504.01697*, 2015. URL <https://arxiv.org/abs/1504.01697>.
- Xiaohan Zhang, Mohamad Altrabulsi, Wenqi Xu, Ralf Wimmer, Michael M. Halassa, and Zhe Sage Chen. Multiplicative couplings facilitate rapid learning and information gating in recurrent neural networks. *bioRxiv*, 2025. 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 \mathbf{O} , input matrix \mathbf{X} and vector coefficient \mathbf{a} , the model representation of the polynomial can be written as:

$$\begin{aligned}
 p(\mathbf{X}) &= \mathbf{X} \mathbf{O} \mathbf{a} = \\
 &= \sum_{d, d'} \mathbf{A}^{d'} \mathbf{O}_{d'}^d \mathbf{X}_d = \\
 &= \sum_{\mathbf{a}, \mathbf{a}'} \left(\sum_{d_1, d'_1} \mathbf{A}_{1a_2}^{d'_1} \mathbf{O}_{1a'_2}^{d'_1 d_1} \tilde{x}_{d_1} \right) \cdots \left(\sum_{d_N, d'_N} \mathbf{A}_{a_N 1}^{d'_N} \mathbf{O}_{a'_N 1}^{d'_N d_N} \tilde{x}_{d_N} \right) = \\
 &= \sum_{\mathbf{a}, \mathbf{a}'} \prod_j \sum_{d_j, d'_j} \mathbf{A}_{a_j a_{j+1}}^{d'_j} \mathbf{O}_{a'_j a'_{j+1}}^{d'_j d_j} \tilde{x}_{d_j}
 \end{aligned} \tag{8}$$

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 equation 8 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 $\mathbf{B}_{r_i r_{i+1}}^{i_i i'_i}$. When the dimension of the rank r is 1 we retrieve linear independent transformations of the inputs.

Transforming the weights of the model as

$$\mathbf{T}_d = \sum_{d'_1} (\mathbf{A}^{(1)})_{d'_1} \mathbf{B}_{d'_1 d_1} \cdots \sum_{d'_N} (\mathbf{A}^{(N)})_{d'_N} \mathbf{B}_{d'_N d_N} \tag{9}$$

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

$$x_p = \sum_k g_k X_{k,p} \quad (10)$$

and the resulting polynomial can be written as:

$$\begin{aligned} p(x_1, \dots, x_P) &= \sum_{\mathbf{p}} \sum_{\mathbf{a}} x_{p_1} A_{1a_2}^{p_1} \dots x_{p_n} A_{a_n 1}^{p_n} = \\ &= \sum_{\mathbf{p}} \sum_{\mathbf{a}} \sum_{k_1} g_{k_1} X_{k_1, p_1} A_{1a_2}^{p_1} \dots \sum_{k_n} g_{k_n} X_{k_n, p_n} A_{a_n 1}^{p_n} = \end{aligned} \quad (11)$$

Note that when the convolution kernels \mathbf{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 equation 11 as an MPO we can reorder the elements, add a summation over a delta function and vectorizing the inputs to obtain

$$\begin{aligned} p(x_1, \dots, x_P) &= \sum_{\mathbf{p}} \sum_{\mathbf{p}'} \sum_{\mathbf{a}} \sum_{k_1} X_{k_1 p_1} g_{k_1} \delta_{p_1 p'_1} A_{1a_2}^{p'_1} \dots \sum_{k_n} X_{k_n p_n} g_{k_n} \delta_{p_n p'_n} A_{a_n 1}^{p'_n} = \\ &= \sum_j \sum_{\mathbf{p}'} \sum_{\mathbf{a}} x_{(k_1, p_1)} G^{(k_1, p_1) p'_1} A_{1a_2}^{p'_1} \dots x_{(k_n, p_n)} G^{(k_n, p_n) p'_n} A_{a_n 1}^{p'_n} \end{aligned} \quad (12)$$

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:

$$G^{(k_n, p_n) p'_n} = g_{k_n} \delta_{p_n p'_n} \quad (13)$$

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

$$G_{a_n a_{n+1}}^{(k_n, p_n) p'_n} = \mathbf{G}_{a_n a_{n+1}}^{k_n} \delta_{p_n p'_n} \quad (14)$$

where now the kernels over different subspaces interact.

A.3 MASKING OPERATOR

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

$$N_{Sym} = \binom{M+D}{M} = \frac{(M+D)!}{D!M!} \quad N_{Asym} \approx M^D \quad (15)$$

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 equation 15.

$$\frac{N_{asym}}{N_{sym}} := K \simeq \left(1 + \frac{n}{d}\right)^{-(d+n+\frac{1}{2})} n^{n+\frac{1}{2}} \sqrt{2\pi} \quad (16)$$

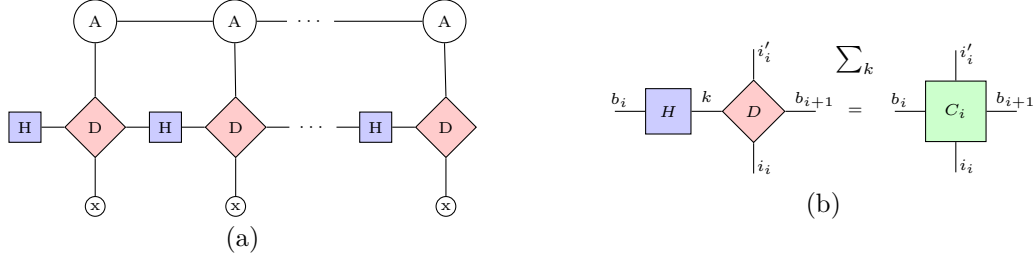


Figure 4: (a) Diagrammatic representation of the masking operator in equation 19. (b) Diagrammatic representation of a block of the masking MPO in equation 21.

for $b, n \gg 1$ (p.s. For $d > 8$ already works). Maintaining n fixed, $\lim_{d \rightarrow \infty} K \simeq e^{-n} n^n$ (which for $n = 6$ is $\simeq 115$ and for $n = 10$ it is $\sim 1/210^5$).

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.

$$y = \sum_{\mathbf{a}} \sum_{d_1=0}^D A_{1a_2}^{d_1} \tilde{x}_{d_1} \sum_{d_2=d_1}^D A_{a_2a_3}^{d_2} \tilde{x}_{d_2} \cdots \sum_{d_n=d_{n-1}}^D A_{a_n1}^{d_n} \tilde{x}_{d_n} \quad (17)$$

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 .

$$\begin{aligned} H_{ij} &= \theta(j - i) \\ D_{ab\dots} &= \delta_{ab\dots} \end{aligned} \quad (18)$$

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:

$$\begin{aligned} p(x_1, \dots, x_n) &= \sum_{\mathbf{a}} \sum_{\mathbf{b}} \sum_{d_1, k, d'_1}^D A_{1a_2}^{d_1} H_{0k} D_{kd_1d'_1b_2} \tilde{x}_{d'_1} \\ &\quad \sum_{d_2, k, d'_2}^D A_{a_2a_3}^{d_2} H_{b_2k} D_{kd_2d'_2b_3} \tilde{x}_{d'_2} \cdots \sum_{d_n, k, d'_n}^D A_{a_n1}^{d_n} H_{b_nk} D_{kd_nd'_nb_n} \tilde{x}_{d'_n} \end{aligned} \quad (19)$$

We can now extract an MPO by defining its blocks.

$$C_{b_i b_{i+1}}^{d_i d'_i} = \sum_k H_{b_i k} D_{kd_i d'_i b_{i+1}} \quad (20)$$

diagrammatically represented in Figure 4

Finally obtaining the masking MPO.

$$C_{\mathbf{d}'}^{\mathbf{d}} = \sum_{b_2, \dots, b_n} C_{1b_2}^{d_1 d'_1} C_{1b_2}^{d_2 d'_2} \cdots C_{b_n 1}^{d_n d'_n} \quad (21)$$

represented in Figure 4.

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.

$$p(x_1, \dots, x_n) = \sum_{d'}^D \sum_d^D A^{d'} C_{d'}^d X_d \quad (22)$$

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

A.4 (MPO)² MODEL ABLATIONS

In Table 3 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.

Table 3: 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 (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.

	SP	AB	OB	BK	RE	EE	CO	AI	PO	SB
(MPO) ² -T1	21.25 ±0.37	20.32 ±22.56	72.02 ±0.75	52.01 ±11.18	79.24 ±0.05	99.75 ±0.00	67.56 ±0.09	42.28 ±0.33	1.81 ±0.21	47.95 ±0.03
(MPO) ² -T2	21.69 ±0.12	58.97 ±0.35	26.07 ±22.39	74.00 ±0.15	80.82 ±0.71	99.78 ±0.01	85.40 ±0.44	37.09 ±1.38	1.58 ±0.21	54.03 ±8.07
(MPO) ² -M-T1	19.74 ±0.26	57.29 ±1.08	72.96 ±0.43	52.50 ±10.92	79.19 ±0.01	99.75 ±0.01	70.46 ±0.83	41.40 ±1.40	1.75 ±0.24	29.54 ±20.93
(MPO) ² -M-T2	21.45 ±0.30	42.31 ±9.56	72.22 ±0.64	74.17 ±0.14	77.79 ±1.18	99.76 ±0.00	85.22 ±0.96	42.45 ±0.37	1.54 ±0.24	60.84 ±2.48
(MPO) ² -L-T1	20.01 ±0.14	48.66 ±7.31	73.53 ±0.55	70.11 ±0.07	79.19 ±0.01	99.75 ±0.01	63.30 ±3.79	41.68 ±0.53	1.96 ±0.24	37.40 ±12.67
(MPO) ² -L-T2	21.67 ±0.14	45.96 ±7.97	60.71 ±14.13	74.23 ±0.13	76.53 ±2.93	99.72 ±0.01	85.31 ±1.50	41.38 ±0.84	1.69 ±0.23	46.40 ±8.63
	IR	HE	WQ	BR	AD	BA	WI	CE	SD	MU
(MPO) ² -T1	95.65 ±0.00	57.83 ±0.87	54.44 ±0.09	97.91 ±0.44	56.87 ±0.02	89.90 ±0.38	100.00 ±0.00	97.69 ±0.21	76.42 ±0.19	99.26 ±0.25
(MPO) ² -T2	100.00 ±0.00	60.43 ±1.74	54.89 ±0.31	99.30 ±0.47	56.30 ±0.45	90.57 ±0.02	97.78 ±0.91	98.46 ±0.17	77.71 ±0.34	99.08 ±0.18
(MPO) ² -M-T1	95.65 ±0.00	57.39 ±1.47	55.30 ±0.65	97.44 ±1.13	56.83 ±0.04	90.39 ±0.03	97.78 ±2.22	97.69 ±0.40	77.11 ±0.38	99.47 ±0.02
(MPO) ² -M-T2	99.13 ±0.87	56.96 ±4.15	54.58 ±0.14	99.77 ±0.23	56.72 ±0.17	89.91 ±0.28	97.78 ±1.48	99.23 ±0.21	76.87 ±1.08	98.93 ±0.23
(MPO) ² -L-T1	95.65 ±0.00	61.74 ±1.11	53.35 ±0.82	96.05 ±0.59	56.79 ±0.03	89.75 ±0.27	100.00 ±0.00	97.92 ±0.36	76.60 ±0.22	99.39 ±0.11
(MPO) ² -L-T2	100.00 ±0.00	53.04 ±4.27	55.08 ±0.34	99.77 ±0.23	56.67 ±0.06	90.26 ±0.28	97.04 ±1.39	99.08 ±0.09	76.93 ±0.86	98.92 ±0.25

A.5 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 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:

$$\mathbf{T}_{i_k}^{(k)} = \begin{cases} \mathbf{0}, & i_k \neq 0 \\ \mathbf{I}, & i_k = 0 \end{cases} \quad \mathbf{T}_{i_0}^{(0)} = \begin{cases} \mathbf{0}, & i_0 \neq 0 \\ \vec{1}, & i_0 = 0 \end{cases} \quad \mathbf{T}_{i_N}^{(N)} = \begin{cases} \mathbf{0}, & i_N \neq 0 \\ \vec{1}, & i_N = 0 \end{cases} \quad (23)$$

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 inequation 23, 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 (MPO)² and CPD against exact regression (learning the full coefficient matrix). Figure 5 reports the average result and confidence interval over twenty randomly generated polynomial, varying number of samples, degree of the model and dimension.

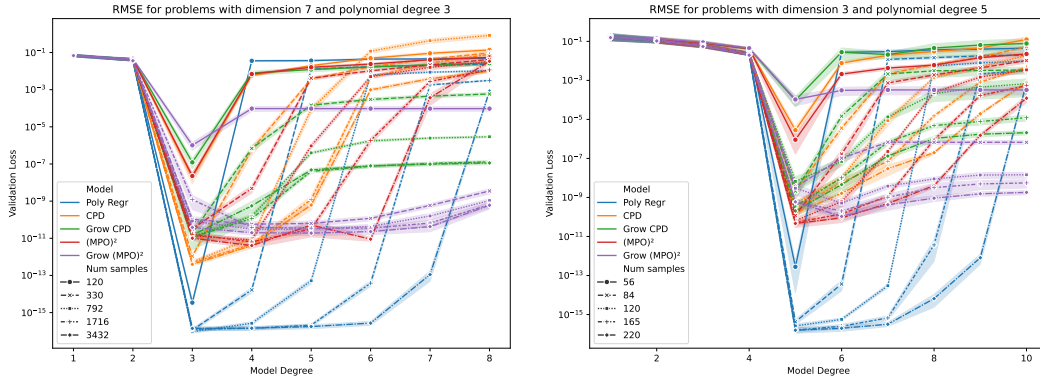


Figure 5: Validation loss across model degrees with sample sizes chosen as the binomial coefficient $\binom{N+D}{D}$, which represent the number of coefficients for a multivariate polynomial of degree N and dimension D .

A.6 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 diabetes¹ due to unavailability for download through the official package. We discarded Automobile² and Auto MPG³ 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 4.

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

A.7 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.7.1 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.

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 4: 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

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.7.2 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 100 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.