Additive Higher-Order Factorization Machines

David Rügamer
david@stat.uni-muenchen.de
LMU Munich; Munich Center for Machine Learning
Munich, Germany

ABSTRACT
In the age of big data and interpretable machine learning, approaches need to work at scale and at the same time allow for a clear mathematical understanding of the method’s inner workings. While there exist inherently interpretable semi-parametric regression techniques for large-scale applications to account for non-linearity in the data, their model complexity is still often restricted. One of the main limitations are missing interactions in these models, which are not included for the sake of better interpretability, but also due to untenable computational costs. To address this shortcoming, we derive a scalable high-order tensor product spline model using a factorization approach. Our method allows to include all (higher-order) interactions of non-linear feature effects while having computational costs proportional to a model without interactions. We prove both theoretically and empirically that our methods scales notably better than existing approaches, derive meaningful penalization schemes and also discuss further theoretical aspects. We finally investigate predictive and estimation performance both with synthetic and real data.

KEYWORDS
smoothing, generalized additive models, scalability

1 INTRODUCTION
Two of the core principles of statistical regression models are additivity and linearity of the predictors. These properties allow estimated feature effects to be easily interpreted, which also led to (re)vised interest in such models in the machine learning and information retrieval community [7, 8, 23, 28, 33, 37, 42, 44, 45]. A frequently used and cited example of an interpretable yet flexible statistical regression model is the generalized additive model [GAM; 15, 39]. Using basis functions to approximate non-linear functions, these models can represent non-linear feature effects in one or a moderate number of dimensions. Applying this principle in settings with many features and higher-order interactions, however, comes with considerable downsides. For univariate non-linear effects, the number of basis functions $M$ for each feature typically lies in the range of 10 to 20 and needs to be evaluated prior to model fitting. Representing and fitting all available features using basis functions will not only result in a notable increase in training time, but also requires a considerable amount of additional memory. In higher dimensions $D$, these problems carry even more weight as $D$-variate non-linear representations are typically constructed using Kronecker or tensor product splines (TPS), i.e., a (row-wise) Kronecker product of all involved bases. This results in computational costs of $O((pM)^D)$ for TPS models with $p$ features. Computational feasibility is thus one of the main reasons statistical applications are often restricted to only uni- and bivariate (tensor product) splines. While several approaches to tackle this problem have been proposed [e.g., 40], existing solutions still either suffer from extensive memory or runtime costs.

Our Contribution: In order to efficiently scale additive models in higher dimensions (cf. also Figure 1), we propose an approach for modeling higher-order TPS with linear complexity in $D$ based on the idea of factorization machines [FMs; 29]. While this effectively addresses existing scaling problems of GAMs, our approach also extends (higher-order) factorization machines by allowing for non-linear relationships. In addition to deriving the resulting computational complexity, we also propose an efficient way of computing the model, suggest a suitable penalization scheme and provide an optimization routine for our approach. Our experimental section demonstrates that models with higher-order TPS work well in practice and yield competitive results in comparison to other commonly used machine learning models.

2 RELATED LITERATURE
Generalized Additive Models and Tensor-Product Splines. GAMs extend generalized linear models [25] by allowing feature effects to be non-linear, typically achieved by using a spline bases representation. Next to basic principles [see, e.g., 15, 39], many extensions of GAMs have been discussed in the literature. In order to include non-linear functions of more than one variable, various options exist, e.g., by using spline bases in multiple dimensions. [38] proposed a flexible way of constructing multivariate non-linear functions in GAMs using TPS as an alternative option, which we will outline.
in more detail in Section 3.2.2. Although GAM software is usually optimized in terms of efficiency, computational costs can be a bottleneck for large data sets or complex model formulations. While there exist approaches work that allows GAM estimation for data with many observations [40], GAMs still scale unfavorably with many features or feature interactions. Recent approaches [31] suggest fitting structured regression models as (part of) a neural network. This can result in a better space complexity in situations with many data points and allows for more flexibility in the additive predictors of models beyond those of classical GAMs [see, e.g., 3, 20].

Factorization Approaches. Similar to GAMs, factorization approaches have been studied extensively. Popularized for recommender systems, different (matrix) factorization approaches have been proposed in the early 2000s [see, e.g., 2, 21, 34] and are still considered state-of-the-art in terms of performance and efficiency [18, 30]. Closely related to matrix factorization are factorization machines [FM; 29]. FM are based on a linear model formulation with pairwise interactions between all features and use a factorization trick to overcome unfavorable scaling when the number of features is large. Various extensions have been developed over the past years, including convex FM [4] and an efficient calculation of higher-order FM [HOFMs; 5]. Other extensions include boosted FM [43], FM with special personalized feature treatment [9] or interaction-aware FM [16]. Similar to our proposal, [22] use a non-parametric subspace feature mapping to encode interactions and account for non-linearity, but rely on binning the features.

Boosting. Apart from FM and GAMs, various other approaches exist that model non-linearity and/or interactions while preserving an additivity structure of the model. One of the most prominent approaches in machine learning that combines additivity and predictive performance is boosting. Already in the seminal work of Friedman [14], boosting was proposed to optimize additive models (potentially with interactions). This idea is used to optimize additive models such as GAMboost [see, e.g., 17] and lays the foundation for other interpretable boosting frameworks such as GA²M [24] and explainable boosting machines [26].

3 BACKGROUND

We first introduce our notation in Section 3.1 and then give a short introduction into GAMs in Section 3.2. For details, see, e.g., [39].

3.1 Notation

In the following, we write scalar values in small or capital letters without formatting, vectors in small bold letters, matrices in capital bold letters, and tensors using fraktur typeface, e.g., \( \mathbf{X} \). Calligraphic letters will have different meaning depending on the context, while \( \mathcal{O} \) is reserved to describe the complexity of calculations in terms of computing time or memory. The Mode-1 fiber of a three-dimensional tensor \( \mathbf{X} \in \mathbb{R}^{P \times P \times P} \) denotes the vectors obtained when fixing the second and third dimension of \( \mathbf{X} \) to certain values \( i, j \), i.e., \( \mathbf{X}_{i,j} \in \mathbb{R}^{P} \). Similar, \( \mathbf{X}_{i,j} \in \mathbb{R}^{P \times P} \) are the frontal slices of \( \mathbf{X} \). For better readability, we will denote the sequence from 1, \ldots, \( \mathbf{x} \) with \( \{\mathbf{x}\} \). We further use \( \mathbf{0} \) for the Kronecker product. For two square matrices \( \mathbf{A}, \mathbf{B} \) with dimensions \( a \) and \( b \), respectively, we define the Kronecker sum as \( \mathbf{A} \oplus \mathbf{B} = \mathbf{A} \otimes \mathbf{I}_b + \mathbf{I}_a \otimes \mathbf{B} \), where \( \mathbf{I}_a \) is the identity matrix of dimension \( \mathbf{x} \). \( \operatorname{vec}(\cdot) \) denotes the vectorization operator to flatten a matrix or tensor along its dimensions.

3.2 Generalized Additive Models

Given the response random variable \( \mathbf{Y} \) and \( p \) features \( \mathbf{x} = (x_1, \ldots, x_p) \), an additive model with linear and non-linear effects for all features assumes the following relationship:

\[
\hat{\mathbf{Y}} = \mathbf{f}(\mathbf{x}) = \mathbf{X} \mathbf{\beta} + \varepsilon = \mathbf{y} + \varepsilon = \alpha_0 + \sum_{j=1}^{p} x_j \alpha_j + \sum_{j=1}^{p} f_j(x_j) + \varepsilon,
\]

where \( \alpha_0, \alpha_1, \ldots, \alpha_p \) are linear regression coefficients, \( f_1, \ldots, f_p \) univariate non-linear functions, \( \varepsilon \sim \mathcal{N}(0, \sigma^2) \) is a zero-mean Gaussian random variable with variance \( \sigma^2 > 0 \) and \( \eta \) the model predictor. GAMs, the generalization of additive models, replace the distribution assumption in (1) using a more general distribution by assuming that \( \mathbf{Y} | \mathbf{x} \) has some exponential family distribution and \( \mathbb{E}(\mathbf{Y}|\mathbf{x}) = h(\eta(\mathbf{x})) \) for some monotonic (response) function \( h \). A prediction \( \hat{\mathbf{y}} = h(\hat{\eta}(\mathbf{x})) \) for the observed value \( \mathbf{y} \) in GAMs is formed by estimating the regression coefficients and functions \( f_j \) GAMs can be optimized using (different types of) maximum likelihood estimation. Alternatively, using the negative log-likelihood as (convex) loss function \( \ell \), their optimization can also be framed as an empirical risk minimization problem.

Since linear effects \( \alpha_j \) can be incorporated in the functions \( f_j \), we will drop the linear model part in the following. The non-linear functions \( f_j \) in GAMs are usually approximated using a (spline) basis representation, i.e.,

\[
f_j(x_j) \approx \sum_{m=1}^{M_j} B_{m,j}(x_j) \beta_{m,j} = B_j^\top \beta_j,
\]

where \( B_{m,j} \) are pre-defined basis functions (e.g., truncated polynomials or B-splines) and \( \beta_{m,j} \) the corresponding basis coefficients.

3.2.1 Smoothness Penalties

One of the most common approaches to estimate smooth functions \( f_j \) is to employ a difference penalty for successive basis coefficients \( \beta_{m,j}, \beta_{m+1,j} \) of basis functions \( B_{m,j}(\mathbf{x}) \), \( B_{m+1,j}(\mathbf{x}) \), which penalizes deviating behavior in neighboring basis functions. The penalty term for the penalized loss function is then given by \( \mathcal{P} = \sum_{j=1}^{p} \lambda_j \int (f_j''(x))^2 \, dx \), which is a trade-off between goodness-of-fit and roughness of the functions \( f_j \). The penalized loss can be written as

\[
\ell(\hat{\mathbf{y}}, \hat{\mathbf{y}}) + \sum_{j=1}^{p} \lambda_j \beta_j^\top P_j \beta_j,
\]

where \( P_j \in \mathbb{R}^{M_j \times M_j} \) is a squared penalty matrix depending on the evaluated basis \( B_j \) for the \( j \)th feature and usually penalizes first or second differences in the coefficients \( \beta_j \).

GAMs also allow for higher dimensional non-linear functions, e.g., bivariate smooth terms \( f_{ij}(x_i, x_j) \). A common approach for their construction are tensor product splines.
3.2.2 Tensor Product Splines. While there are various approaches to construct smooth functions of several features, tensor product splines (TPS) constructed from marginal univariate bases constitute an attractive option. The resulting smooth terms are very flexible, scale-invariant, relatively low rank as well as easy to construct and interpret [see 38]. For a model with all \( \binom{p}{2} \) possible bivariate effects, the TPS part is given by

\[
\sum_{k=1}^{p} \sum_{l=k+1}^{p} f_{k,l}(x_k, x_l) \approx \sum_{k=1}^{p} \sum_{l=k+1}^{p} (B_k \otimes B_l) \beta_{k,l}
\]

with univariate spline basis functions \( B_{m,k} \) and \( B_{o,l} \) and basis coefficients \( \beta_{m,k,o,l} \), summarized in \( \beta_{k,l} \in \mathbb{R}^{M_k\times O_l} \). Bivariate TPS are penalized using

\[
\mathcal{P}(f_{k,l}) = \int_{x_k,x_l} \lambda_k (\partial^2 f / \partial x_k^2)^2 + \lambda_l (\partial^2 f / \partial x_l^2)^2 \, dx_k dx_l,
\]

which can be written as

\[
\mathcal{P}(f_{k,l}) = B_{k,l}^\top (\lambda_k P_k \otimes \lambda_l P_l) \beta_{k,l}.
\]

This principle can be generalized to \( D \)-variate smooths for variables \( \mathcal{J} = \{j_1, \ldots, j_D\} \), which are approximated by

\[
f_{j_1,\ldots,j_D}(x_{j_1}, \ldots, x_{j_D}) = (\otimes_{j \in \mathcal{J}} B_j) \beta_{j,j}.
\]

while this seems inconspicuous at first glance, a naive approach for bivariate models requires storing \( O(p^2 M^2) \) entries, which is infeasible if \( n \) or \( p \) is large. In contrast, AFMs only require the same amount of storage as for a univariate spline model. Note that the number of parameters also (linearly) increases with \( F \) and needs to be taken into account for the total required storage. However, Proposition 4.3 specifically looks at the costs of storing basis evaluated features in memory as this can be a storage bottleneck during the pre-processing of GAMs.

4 SCALABLE HIGHER-ORDER TENSOR PRODUCT SPLINE MODELS

As can be directly inferred from (4), the cost of fitting a bivariate TPS is \( O(p^2 M^2) \). While \( M \) is usually kept fixed and of moderate size (e.g., \( M = 10 \)), the function models will be increasingly expensive for both a growing number of basis evaluations and number of features \( p \). For models with (up to) \( D \)-variate TPS, the computational cost increases to \( O(p^D M^D) \). This makes GAMS infeasible both in terms of computing time and also in terms of memory storage.

4.1 Additive Factorization Machines

To overcome the unfavorable scaling of GAMs with many (or higher-order) TPS, we introduce additive factorization machines (AFMs). Based on the idea of factorization machines, we approximate \( f_{k,l}(x_k, x_l) \) in (4) by \( \psi_{k,l}(x_k, x_l) \) defined as

\[
\sum_{m=1}^{M_k} \sum_{o=1}^{O_l} B_{m,k}(x_k) B_{o,l}(x_l) \psi_{m,k,o,l,1},
\]

where \( \gamma_{m,k,o,l} \in \mathbb{R} \) are latent factors approximating the joint effect \( \beta_{m,k,o,l} \). When approximating every bivariate interaction term in (4) with the term defined in (8), we can derive the following representation.

Corollary 4.1 (AFM Representation). The approximation of (8) using (4) can be written as

\[
\sum_{k=1}^{p} \sum_{l=k+1}^{p} \psi_{k,l}(x_k, x_l) = \frac{1}{2} \sum_{f=1}^{F} \left( \sum_{k=1}^{p} \psi_{k,f}^2 - \sum_{k=1}^{p} \psi_{k,f}^2 \right),
\]

with \( \psi_{k,f} = \sum_{m=1}^{M_k} B_{m,k}(x_k) \gamma_{m,k,f} \).

As a direct result of Corollary 4.1, we obtain the scaling of computing AFMs.

Proposition 4.2 (Linear Scaling of AFMs). Computations for AFMs scale with \( O(pM) \).

Corollary 4.1 and Proposition 4.2 are natural extensions of linearity results from FMs. A proof of the corollary is provided in the Appendix. Roughly speaking, the factorization trick from FMs also works for AFMs in a similar manner as the additional basis function dimension only depends on the respective feature dimension. In particular, this means that AFMs scale linearly both in the number of features \( p \) and the spline basis dimension \( M \). Another direct result of this representation and noteworthy property unique to AFMs is given in the following proposition for a dataset of \( n \) observations.

Proposition 4.3 (Basis Evaluations in AFMs). If every feature in AFMs is represented by only one basis, it suffices to evaluate all univariate basis functions once for each feature and the memory costs for storing all features are \( O(npM) \).

While this seems inconspicuous at first glance, a naive approach for bivariate models requires storing \( O(p^2 M^2) \) entries, which is infeasible if \( n \) or \( p \) is large. In contrast, AFMs only require the same amount of storage as for a univariate spline model. Note that the number of parameters also (linearly) increases with \( F \) and needs to be taken into account for the total required storage. However, Proposition 4.3 specifically looks at the costs of storing basis evaluated features in memory as this can be a storage bottleneck during the pre-processing of GAMs.
only the last term in (10) is already \(O(p^DM^D)\) (and analogous for memory costs).

Inspired by Vieta’s formula and the ANOVA kernel, we can derive a similar result as given in [5] to reduce the cost of computing the \(d\)th degree term in AHOFMs. We will make the degree \(d\) explicit for \(\gamma\) and \(\nu\) using the superscript \((d)\).

**Definition 4.4** (Additive Higher-order Term (AHOT)). The \(f\)th additive higher-order term (AHOT) of degree \(d \leq d \leq D\) in AHOFMs is given by

\[
\phi_f^{(d)} = \sum_{j_d \geq \cdots \geq j_1} \prod_{t=1}^{d} M_{j_t} B_{m_{j_t}}(x_{j_t}) y_{m_{j_t}, f}^{(d)}.
\]

(11)

We use \(F_d\) AHOTs to approximate a \(d\)-variate smooth:

\[
\sum_{f=1}^{F_d} \phi_f^{(d)} \approx \sum_{j_d \geq \cdots \geq j_1} f_{j_1, \ldots, j_d}(x_{j_1}, \ldots, x_{j_d}).
\]

(12)

and estimate the \(D\)-variate TPS model (10) with an AHOFM of degree \(D\), defined as follows.

**Definition 4.5** (AHOFM of Degree \(D\)). The predictor \(\eta(x)\) of an AHOFM of degree \(D\) is defined by

\[
\eta(x) = \sum_{j=1}^{P} B_{m_{j}}(x_{j}) \beta_{m_{j}} + \frac{D}{d} \sum_{d=1}^{D} \sum_{f=1}^{F_d} \phi_f^{(d)}.
\]

(13)

The following corollary defines how to recursively describe all AHOTs for \(d \geq 2\) using univariate spline representations \(\varphi_{j,f}\) as defined in (9).

**Lemma 4.6** (Representation AHOT of Degree \(d\)). Let \(\phi_f^{(0)} \equiv 1\) as well as \(\phi_f^{(1)} = \sum_{j=1}^{P} \varphi_{j,f}\). The degree \(d \leq 2\) AHOT can be recursively defined by

\[
\phi_f^{(d)} = \frac{1}{d} \sum_{t=1}^{d} (-1)^{t+1} \phi_f^{(d-t)} \left( \sum_{j=1}^{P} \varphi_{j,f}^{(d-t)} \right).
\]

(14)

The recursive representation (14) allows us to efficiently calculate AHOTs of higher order. The corresponding proof can be found in the Appendix. As another consequence of this representation, we have the following scaling properties.

**Proposition 4.7** (Linear Scaling of AHOFMs). Computations for AHOFMs scale with \(O(pMF D + FD^2)\) where \(F = \sum_{d=1}^{D} F_d\).

Since \(D\) is usually small, computations again roughly scale linearly with the number of features, the basis, and the latent factor dimension. We also recognize that despite the increased dimension \(D\), every feature basis has to be evaluated only once.

**Proposition 4.8** (Basis Evaluations in AHOFMs). If every feature in AHOFMs is represented by only one basis, it suffices to evaluate all univariate basis functions once for each feature and the memory costs for storing all features are \(O(npM)\).

While the memory consumption also increases with \(F\) when considering the storage of all \(\gamma\) parameters, Proposition 4.8 again focuses on the storage of all features after applying the basis evaluations. Similar to the kernel trick, higher-order features are not actually calculated and stored in memory, as AHOFMs only work on the (basis evaluated) univariate features instead of \(\gamma\).

In contrast to FMs and HOFMs, AHOFMs require additional considerations to enforce appropriate smoothness of all non-linear functions in (10) without impairing favorable scaling properties.

### 4.3 Penalization and Optimization

An additional challenge in learning many, potential higher-order TPS, is their optimization in terms of appropriate smoothness.

#### 4.3.1 Penalization

Following the penalization scheme of TPS described in Section 3.2.2, we propose a smoothness penalization for AHOFMs based on the penalties of involved marginal bases \(B_{m_{j}}\) with corresponding difference penalty matrices \(P_f\). Let \(\theta_f^{(d)}\) be the array of all coefficients \(y_{m_{j}, f}^{(d)}\) for all \(m \in \{M\}, j \in \{p\}, f \in \{F_d\}\). For simplicity, we assume \(M_j \equiv M\), so that \(\theta_f^{(d)} \in \mathbb{R}^{M \times p \times F_d}\). Further, let \(\theta_f^{(D)} = \theta_f^{(1)}, \ldots, \theta_f^{(D)}\) and \(y_{j,f}^{(d)} = (y_{1, j, f}^{(d)}, \ldots, y_{M_j, f}^{(d)})^T\), i.e., the Mode-1 (column) fibers of \(\theta_f^{(d)}\), and \(\Theta = (\alpha, \beta_1, \ldots, \beta_p)\).

We define the penalty of AHOFMs as follows.

**Definition 4.9** (AHOFM Penalty). The smoothing penalty of AHOFMs is defined as

\[
\mathcal{P}(\theta_f^{(D)}, \Theta) = \sum_{j=1}^{P} \lambda_j \beta_j \frac{1}{2} \sum_{d=2}^{D} \sum_{f=1}^{F_d} \lambda_f^{(d-1)} \sum_{j=1}^{P} \sum_{\gamma_j}^{P} \gamma_j(f)^T y_{j,f}^{(d)} P_f y_{j,f}^{(d)} + \frac{1}{2} \sum_{f=1}^{F_d} \sum_{\gamma_j}^{P} \gamma_j(f)^T P_f y_{j,f}^{(d)} + \frac{1}{2} \sum_{f=1}^{F_d} \sum_{\gamma_j}^{P} \gamma_j(f)^T P_f y_{j,f}^{(d)} + \frac{1}{2} \sum_{f=1}^{F_d} \sum_{\gamma_j}^{P} \gamma_j(f)^T P_f y_{j,f}^{(d)} + \frac{1}{2} \sum_{f=1}^{F_d} \sum_{\gamma_j}^{P} \gamma_j(f)^T P_f y_{j,f}^{(d)} + \frac{1}{2} \sum_{f=1}^{F_d} \sum_{\gamma_j}^{P} \gamma_j(f)^T P_f y_{j,f}^{(d)}.
\]

(15)

Due to the independence assumption of all latent factors involved in every factorization, it is natural to only penalize univariate directions as expressed in Definition 4.9. A regularization that involves multiple dimensions would further result in a non-decomposable penalty w.r.t. the \(y_{j,f}^{(d)}\) and make the optimization of AHOFMs more challenging (see Section 4.3.3 for details). The penalized optimization problem for \(n\) i.i.d. data points \((y_i, x_i) \in [n]\) with \(x_i = (x_{i,1}, \ldots, x_{i,p})\) and loss function \(f\) is then given by

\[
\arg\min_{\theta_f^{(D)}, \Theta} \sum_{i=1}^{n} f(y_i, \eta(x_i)) + \frac{1}{2} \mathcal{P}(\theta_f^{(D)}, \Theta).
\]

(16)

In (16), the smoothing parameters are considered to be tuning parameters. While it is possible in univariate GAMs to estimate the smoothing parameters \(\lambda_1, \ldots, \lambda_p\) directly, this becomes computationally challenging for higher-order TPS due to the exponentially increasing amount of parameters. HOFMs avoid this combinatorial explosion of hyperparameters by setting all the parameters to the same value [5]. This, however, is not a meaningful approach for smoothing parameters as every smooth term can potentially live on a completely different domain (e.g., \(\lambda = 1\) can imply no penalization for one smooth, but maximum penalization for another term).

#### 4.3.2 Scalable Smoothing

To derive a meaningful penalization in AHOFMs, we exploit the definition of degrees-of-freedom for penalized linear smoothers [6]. Given a matrix of basis evaluations \(B_f \in \mathbb{R}^{n \times M_j}\) with entries \(B_{m_{j}}(x_{i,j})\) and a squared penalty matrix \(P_f\) for the penalization of differences in neighboring basis coefficients,

---

1Next to a ragged tensor definition that allows for a varying first dimension, padding the tensor can also be an option to always have \(M\) dimensions for every feature \(j\).
there exists a one-to-one map between $\lambda_{j,f}^{(d)}$ and the respective degrees-of-freedom

$$d\lambda_{j,f}^{(d)}(\lambda_{j,f}^{(d)}) = \text{tr}(2H_j(\lambda_{j,f}^{(d)}) - H_j(\lambda_{j,f}^{(d)})^\top H_j(\lambda_{j,f}^{(d)}))$$  \hspace{1cm} (17)

with $H_j(\lambda_{j,f}^{(d)}) = B_j(B_j^\top B_j + \lambda_{j,f}^{(d)}I_p)^{-1}B_j^\top$. While the exact degrees-of-freedom only hold for a linear model with a single smooth term, this approach allows to define a meaningful a priori amount of penalization for all smooth terms by restricting their degrees-of-freedom to the same global $d\lambda_{j,f}^{(d)}$ value as follows.

**Proposition 4.10** (Homogeneous AHOFM Smoothing). Given a global $d\lambda_{j,f}^{(d)}$ value, an equal amount of penalization for all $F\lambda_{j,f}^{(d)}$ AHOTs in $\Phi_f^{(d)}$ is achieved by choosing $\lambda_{j,f}^{(d)}$ such that $d\lambda_{j,f}^{(d)}(\lambda_{j,f}^{(d)}) = d\lambda_{j,f}^{(d)} \forall j \in [p], f \in [F_d]$.

The Demmler-Reinsch Orthogonalization [DRO; 32] can be used to efficiently solve (17) for $\lambda_{j,f}^{(d)}$, i.e., calculate $\lambda_{j,f}^{(d)}$ based on a given value $d\lambda_{j,f}^{(d)}$. The DRO involves the calculation of singular values $s_j$ of a squared $M_j \times M_j$ matrix. Once $s_j$ are computed, (17) can also be solved multiple times for different $d\lambda_{j,f}^{(d)}$ values without additional costs. More details are given in Appendix C.1. Moreover, as the factorization only requires univariate smooth terms, $\lambda_{j,f}^{(d)}$ can be calculated for every feature separately at the cost of $O(M_j^3)$ due to our factorization approach. This cost is comparatively small compared to a computation for all features in a $D$-ivariate interaction term with $O(\lambda_3^{3D})$. Also note that in Proposition 4.10, $d\lambda_{j,f}^{(d)}(\lambda_{j,f}^{(d)})$ only needs to be calculated once for every $j$ as all involved matrices in (17) are independent of $f$, and can be done prior to the optimization with no additional costs during training. Algorithm 4 summarizes the routine. Homogeneous AHOFM smoothing amounts to equally flexible non-linear interactions for every order-$d$ AHOT and hence implies isotropic smoothing for all TPS. Given no a priori information on the non-linear interactions of all features, this is a natural choice. In contrast, if we choose different values for one or more features, i.e., $\exists j : df_{j,f} \neq df$, all TPS involving the $j$th feature are subject to anisotropic smoothing.

4.3.3 Optimization. In order to scale also for large numbers of observations, we propose a stochastic mini-batch gradient descent routine for the optimization of A(HO)FMs. We discuss the optimization problem in Appendix B and suggest a block-coordinate descent (BCD) as an alternative optimization routine by showing that the problem in (16) is coordinate-wise convex in $\Phi_f^{(d)}$ (Lemma B.1). In practice, however, different BCD variants showed slow convergence and finding a good choice for hyperparameters such as the learning rate proved to be challenging. In contrast, sophisticated stochastic gradient descent routines such as Adam [19] showed similar or even better results (for a given limited time budget).

5 NUMERICAL EXPERIMENTS

In the following, we will empirically investigate the performance of A(HO)FMs using simulation and a benchmark studies.

---

**Figure 2:** Example of estimated and true surfaces for the different TP splines (columns) and different methods (different rows) visualized by contour plots. Colors represent the partial effect value.

---

5.1 Estimation Performance

We first compare the estimation performance of our proposal with the SotA for fitting GAMs with TPS. More specifically, we simulate features and generate bivariate non-linear effects for every possible feature pair. The response is generated by adding random noise with a signal-to-noise ratio (SNR) of 0.5 to the sum of all bivariate effects. We then compare the estimation performance of all feature effects qualitatively by inspecting the estimated non-linear effects visually (cf. Figure 2 for $n = 2000$), and quantitatively by computing the mean squared error (MSE) between the estimated and true surfaces (Figure 3). For $n \in \{2000, 4000, 8000\}$ and $p = 5$ (resulting in 10 bivariate effects), we run AFMs with $F \in \{1, 5, 15\}$. We repeat every setting 10 times with different random seeds. The GAM estimation can be thought of as a gold standard which is only subject to an estimation error, but no approximation error. In contrast, AFMs are also subject to an approximation error. Our quantitative analysis of results confirms this hypothesis. Figure 3 depicts the MSE differences for all analyzed settings to compare the estimation performance of AFMs and GAMs when calculating the average point-wise differences between the estimated bivariate surface and the true surface. Results suggest that for increasing $F$ our approach will approach the estimation performance of GAMs. With more data (larger $n$), this effect becomes even more

5.2 Prediction Performance

To investigate the prediction performance, we follow the setup from the previous section and compare the prediction performance of the exact GAM and our approach to quantify the approximation error made by the factorization. Figure 4 depicts the results, confirming that the approximation error will tend to zero when increasing the number of observations $n$ or latent factors $F$. 
Figure 3: Estimation quality measures by the MSE difference between a GAM estimation and our proposal with different numbers of latent dimensions $F$ (x-axis) and different numbers of observations (columns). Points correspond to different simulation replications and surfaces. A blue smoother function visualizes the trend in $F$.

5.3 Scalability

Our next experiment investigates the scaling behavior of our approach and compares it to the SotA implementation for big additive models [BAMs 40]. We simulate $p \in \{3, 6, 9, 12\}$ standard normal distributed features for $n \in \{6000, 12000, 18000\}$ observations and fit both BAM and an AFM to learn a GAM with TP splines for all possible combinations of the $p$ features. Figure 1 (first page) summarizes the results by comparing the memory consumption (in megabytes) and the computing time (in seconds). The results reflect our initial motivation to propose AFMs. While both time and memory consumption grows exponentially in the number of features for BAMs, we observe a linear scaling for AFMs both for memory consumption and computation time.

5.4 Ablation Benchmark Study

To assess the prediction performance of AFMs and AHOFMs on real-world data and better understand their advantages but also their limitations, we compare both approaches against a variety of alternatives with similar properties. In particular, all methods use linear or spline feature effects (i.e., we do not compare against non-additive or tree-based methods), and instead of presenting a benchmark where the proposed methods excel for all data sets, we present various scenarios that allow us to objectively reason about the methods’ pros and cons. More specifically, we compare our methods against GAMs with only univariate smooth terms, FMs, HOFMs, as well as boosting with linear effects (GAMBoost) and boosting with splines [GAMBoost; 17]. Whereas GAMs and GLM-/GAMBoost are tuning-free methods, all factorization approaches are analyzed for only three different numbers of latent dimensions ($F \in \{1, 5, 15\}$) to provide a more fair comparison. We compare all methods on commonly used benchmark data sets using 10 train-test splits and report average MSE values as well as their standard deviation. Table 1 summarizes the results when choosing the best-performing hyperparameter set per method and data set. Further details on hyperparameters and benchmark data sets can be found in Section E.2 in the Appendix. From our results, the following research hypotheses can be derived: Prediction performance can often be improved by including

- interactions (GAMs vs. A(HO)FMs);
- non-linearity ((HO)FMs vs. A(HO)FMs);
- higher-order interactions (AFM vs. A(HO)FMs).

We can also identify weaknesses of A(HO)FMs. Feature selection (provided by GLM-/GAMBoost but not A(HO)FMs) and inhomogeneous smoothing (GAMBoost) can improve prediction performance. These two aspects are thus a promising direction for future research.

6 SUMMARY AND OUTLOOK

We presented an additive model extension of HOFMs to allow for scalable higher-order smooth function estimation based on tensor product splines. The proposed approach allows fitting GAMs with $D$-variate smooth terms at costs similar to a univariate GAM. Our simulation studies showed that when choosing enough latent factors, these machines approximate TPS surfaces very well and match the (almost) exact GAM with TPS both in terms of estimation and prediction performance. A(HO)FMs thereby not only allow to fit higher-order GAMs, but also make additive models more competitive in their predictive performance. This was shown in our benchmark study on real-world data, where the true data-generating process is not necessarily a GAM. Here, AHOFMs improved over the GAM and (HO)FM prediction performance due to the inclusion of interactions and extension to non-linearity. A promising future research direction is the sparsification of the proposed approach. This is also closely related to the interpretability of A(HO)FMs, which we discuss in Appendix D in more detail.
### A PROOFS

#### A.1 Proof of Corollary 4.1

For simplicity assume that the model predictor only consists of a single AHOT, i.e., \( \eta(x) = \sum_{j=1}^{F_d} \Phi_f^d \). The generalization of the following statement to several AHOTs follows due to the additivity of the model predictor. Using Equation (20) and constants \( \xi_f, \zeta_f \), it follows

\[
\eta(x) = \sum_{f=1}^{F_d} \Phi_f^d = \sum_{f=1}^{F_d} \left( \xi_f + \eta_f \right) = \sum_{f=1}^{F_d} \left\{ \xi_f + \sum_{j=1}^{F_d} \Phi_f^d \right\} = \sum_{f=1}^{F_d} \left\{ \xi_f + \sum_{j=1}^{F_d} \Phi_f^d \right\} = \sum_{f=1}^{F_d} \left\{ \xi_f + \eta_f \right\}
\]

#### A.2 Proof of Proposition 4.2 and 4.3

Both propositions directly follow from the fact that (18) only sums over \( f, k \) and \( m \) once, and every basis function \( B_{m,k} \) is therefore also only evaluated once.

#### A.3 Proof of Lemma 4.6

An alternative representation of (4.6) is given by

\[
\psi_f^d = \sum_{j=1}^{F_d} \psi_{f,j}
\]

by just plugging in the definition for \( \varphi \). We can consider (19) as an ANOVA kernel of degree \( d \) in the new feature space given by all \( \varphi \). As a result, the multi-linearity property of the ANOVA kernel holds [see 5, Appendix B.1], i.e.,

\[
\psi_f^d = \psi_{f,j} + \varphi_{f,j} \psi_j^{d-1}
\]

where \( \psi_{f,j} = \sum_{j=1}^{F_d} \Phi_f \) and therefore AHOTMs can be represented as in Lemma 4.6 by using the same arguments as for HOFMs [5].

#### A.4 Proof of Lemma B.1

For simplicity assume that the model predictor only consists of a single AHOT, i.e., \( \eta(x) = \sum_{j=1}^{F_d} \Phi_f^d \). The generalization of the following statement to several AHOTs follows due to the additivity of the model predictor. Using Equation (20) and constants \( \xi_f, \zeta_f \), it follows

\[
\eta(x) = \sum_{f=1}^{F_d} \Phi_f^d = \sum_{f=1}^{F_d} \left( \xi_f + \eta_f \right) = \sum_{f=1}^{F_d} \left\{ \xi_f + \sum_{j=1}^{F_d} \Phi_f^d \right\} = \sum_{f=1}^{F_d} \left\{ \xi_f + \sum_{j=1}^{F_d} \Phi_f^d \right\} = \sum_{f=1}^{F_d} \left\{ \xi_f + \eta_f \right\}
\]

where

\[
\eta_f = \sum_{j=1}^{F_d} \Phi_f^d = \sum_{f=1}^{F_d} \left( \xi_f + \eta_f \right) = \sum_{f=1}^{F_d} \left\{ \xi_f + \sum_{j=1}^{F_d} \Phi_f^d \right\} = \sum_{f=1}^{F_d} \left\{ \xi_f + \sum_{j=1}^{F_d} \Phi_f^d \right\} = \sum_{f=1}^{F_d} \left\{ \xi_f + \eta_f \right\}
\]

and hence also in \( \Gamma_f^d \), the penalized objective in (16) is convex w.r.t. every \( \Gamma_f^d \) and thus every \( \gamma_{j,f} \) in [p], \( f \in [F_d] \).

### B OPTIMIZATION

Having defined the objective in (16), we obtain the following result.

**Lemma B.1.** The optimization problem in (16) is coordinate-wise convex in \( \gamma_{j,f}^d \).
A corresponding proof is given in Appendix A. Using this finding suggests a block coordinate descent (BCD) solver as an alternative approach to optimize (penalized) AHOFMs with block updates for \( y_{j,f}^{(d)} \). In contrast to (H)OFMs, we perform block updates instead of plain coordinate descent as the AHOFM penalty is only decomposable w.r.t. all \( y_{j,f}^{(d)} \), but not w.r.t. \( y_{m,j,f}^{(d)} \). For BCD we require several quantities:

\[
\nabla \varphi_{j,f}(x_{i,j}) := \partial \varphi_{j,f}(x_{i,j})/\partial y_{j,f} = B_j(x_{i,j}),
\]

\[
\nabla \Phi_{j,f}^{(d)}(x_i) := \partial \Phi_{j,f}^{(d)}(x_i) / \partial y_{j,f}^{(d)} = \frac{1}{d} \sum_{t=1}^d (-1)^{t+1} \left\{ \frac{\partial \Phi_{j,f}^{(d-t)}(x_i)}{\partial y_{j,f}^{(d)}} \sum_{j=1}^p \phi_{j,f}^{(d)}(x_{i,j}) \right\}^t + \phi_{j,f}^{(d-t)}(x_i) \sum_{j=1}^p \phi_{j,f}^{(d)}(x_{i,j})^{-1} \nabla \varphi_{j,f}(x_{i,j}),
\]

\[
\nu = \sum_{i=1}^n \partial^2 \ell(x_i, y_i) = \partial (\varphi_{j,f}^{(d)})^2 + \lambda_{j,f} P_j \nu_{j,f}^{(d)},
\]

\[
\frac{\partial \mathcal{L}(x_i, y_i)}{\partial y_{j,f}^{(d)}} = \frac{\partial \ell(x_i, y_i)}{\partial y_{j,f}^{(d)}} = \nabla \Phi_{j,f}^{(d)}(x_i) + \lambda_{j,f} P_j \nu_{j,f}^{(d)},
\]

(22)

Note that the first term is involved in every update step, but independent of \( f \) and the iteration. It is therefore possible to cache the result once at the beginning of the training routine as also mentioned in Algorithm 5. For reverse-mode differentiation, note that the second term can be calculated efficiently by caching intermediate results.

Algorithm 1 High-level routine is described in Algorithm 1 (for simplicity for the case with only a single \( D \)-variate smooth). In Algorithm 1, we require gradients

\[
\nabla \mathcal{L}(y_{j,f}^{(d)}) := \sum_{i=1}^n \frac{\partial \mathcal{L}(x_i, y_i)}{\partial y_{j,f}^{(d)}},
\]

where \( \mathcal{L} \) is the objective function from (16) and a learning rate \( \nu \), which is defined above. Various terms involved in the update step can be pre-computed or cached (see Appendix C for details).

Algorithm 1 BCD AHOFM Optimization

Input: Data \((y_i, x_{i})_{i \in [n]}\); \( B_{m,j}, P_j, \forall j \in [p], m \in [M_j]; D; \); \( \text{df}^{(D)} \); \( F_D \); BCD convergence criterion
Initialization: \( \tilde{\nu}, \tilde{G}^{(D)} = \text{init}(\text{Input}) \) (Appendix C.3)
repeat
  for \( f = 1 \) to \( F_D \) do
    for \( j = 1 \) to \( p \) do
      Calculate step-size \( \nu \)
      Update \( y_{j,f}^{(d)} \leftarrow y_{j,f}^{(d)} - \nu \nabla \mathcal{L}(y_{j,f}^{(d)}) \)
      Synchronize \( \tilde{\nu}, i \in [n] \)
  end for
end for
until convergence

C ALGORITHMIC DETAILS

C.1 Demmler-Reinsch Orthogonalization

We here describe the DRO (Algorithm 2) and sv2la (Algorithm 3) routine proposed to efficiently compute smoothing penalties. Details can be found in [32], Appendix B.1.1. We use Chol to denote the Cholesky decomposition of a matrix and SVD for the singular value decomposition of a matrix.

Algorithm 2 DRO

Input: Feature matrix \( B \in \mathbb{R}^{n \times M} \), penalty matrix \( P \in \mathbb{R}^{M \times M} \)
Compute:
1. \( R^T R \leftarrow \text{Chol}(B^T B) \)
2. \( U \text{diag}(s) U^T \leftarrow \text{SVD}(R^T P R^{-1}) \)
Output: singular values \( s \)

Algorithm 3 sv2la

Input: Singular values \( s \in \mathbb{R}^M, \text{df} \)
Define \( \text{diffun}(\lambda) = \sum_{j=1}^M (1 + I s_j)^{-1}; \)
Compute: \( \lambda \) for which \( \text{diffun}(\lambda) = \text{df} \) using a uniroot search;
Output: \( \lambda \)

C.2 Homogeneous AHOFM Smoothing

Given the previous algorithms, we can implement homogeneous AHOFM smoothing as described in Algorithm 4.

Algorithm 4 Homogeneous AHOFM Smoothing

Input: \( B_j, P_j \forall j \in [p]; \text{df}^{(d)} \forall d \in [D] \)
for \( j = 1 \) to \( p \) do
  Compute \( s_j = \text{DRO}(B_j, P_j) \) (costs: \( O(M_j^2) \))
  Compute \( \lambda_{j,f}^{(d)} = \text{sv2la}(s_j, \text{df}^{(d)}) \) (negligible costs);
  Set \( \lambda_{j,f}^{(d)} = \lambda_{j,f}^{(d)} \) for \( f \in [F_d] \);
end for
Output: \( \lambda_{j,f}^{(d)} \) for all \( j \in [p], d \in [D], f \in [F_d] \)

C.3 AHOFM Initialization

Putting everything together, the initialization of AHOFMs is given in Algorithm 5.

D INTERPRETABILITY OF A(HO)FMS

Due to their additivity assumption, every additive feature effect in GAMs can be interpreted on its own (ceteris paribus). Although AHOFMs inherit some of the interpretability properties from GAMs, e.g., their additivity, interpreting higher-order (non-linear) interaction terms remains challenging and cannot be done without considering lower-order effects of the same feature. For larger values of \( p \), the quickly growing number of additive terms further makes

\[
\text{Algorithm 4.}
\]

\[
\text{Algorithm 5.}
\]
it infeasible to grasp the influence of certain features or interactions. While this is a limitation of the current approach, we here propose three ways to check effects for models with small to moderate \( p \). The first approach examines interaction terms by visualizing the single univariate smooth terms \( \phi_{j,f}(x) \) for \( f = 1, \ldots, p \) and all involved feature dimensions \( j \). Analyzing the univariate latent dimensions separately is not a new approach and the use of factorization approaches can even be motivated by the need to interpret higher-dimensional interactions in lower dimensions [see, e.g., 35]. Another approach that focuses only on the interaction effects itself is to visualize the actual approximations \( \hat{f}_{j,f}(x) \) as defined in (8). This reduces the number of terms to analyze by the factor \( F_d \), but requires a method for presenting the \( d \)-variate effect. As shown in Figure 5, a third approach is to visualize the marginals of these multivariate functions together with their variation across the respective other dimensions.

**Experiments.** We simulate a toy example for \( D = 3 \) with four features to demonstrate the third approach. For better understanding, the features are referred to as `time`, `lat`, `lon` and `rate`. The outcome is assumed to be normally distributed with \( \sigma = 0.1 \) and the mean given by the sum of all possible smooth 3-way interactions of the features. To simulate non-linear three-dimensional functions, we use a basis evaluation of features with 4 degrees-of-freedom for `time`, 5 degrees-of-freedom for `lat`, 7 degrees-of-freedom for `lon` and 5 degrees-of-freedom for `rate`. Partial effects for each three-dimensional smooth are generated by calculating the TP for these basis and randomly drawing coefficients for the resulting TPS. We generate \( 10^5 \) observations, fit a \( \text{AHOFM}(D = 3) \) and visualize the resulting effects in two different ways. Figure 5 depicts the marginal univariate effects of all four features for all four 3-way interaction effects. This allows us to see how each feature marginally affects each of the interaction terms. Additionally, it shows how much variation the marginal effects have in the respective other two dimensions and thereby provides information on how much the features interact with the respective two other variables.

**E NUMERICAL EXPERIMENTS DETAILS**

**E.1 Implementation**

All methods have been implemented in TensorFlow [1] except for GAMBoost, where we used the package mboost [17] and the SotA GAM, implemented in mgcv [39].

**Algorithm 5 AHOFM Init**

**Input:** Data \((y_i, x_i), i \in [n]\); order \( D \), bases functions \( B_{m,j}, P_j, j \in [p], m \in [M_j]; d_f^{(d)} \)

Compute the following quantities:

- \( y_{j,f}^{(d)} \forall j \in [p], f \in [F_d] \) using Algorithm 4;
- \( \nabla \phi_{j,f}(x) \forall i \in [n] \) as in (22);

Cache the derivatives \( \nabla \phi_{j,f}(x) \) for later update steps;

Randomly initialize \( y_{j,f}^{(d)} \) and calculate \( \hat{y}_{j,f}^{(d)} \) for all \( j \in [p], f \in [F_d] \);

Compute \( \hat{\eta} \)

**Output:** \( \hat{\eta}, \theta_{f}^{(d)} \)

**REFERENCES**

[1] Martin Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. 2016. Tensorflow: A system for large-scale machine learning. In 12th (USENIX) symposium on operating systems design and implementation (OSDI’16) 265–283.

[2] Gediminas Adomavicius and Alexander Tuzhilin. 2005. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. IEEE transactions on knowledge and data engineering 17, 6 (2005), 734–749.
