SAND2018-0992R

Gradient-based Optimization for Regression in the Functional Tensor-Train Format

A.A Gorodetsky*†, J.D Jakeman‡

December 6, 2017

Abstract

We consider the task of low-multilinear-rank functional regression, i.e., learning a low-rank parametric representation of functions from scattered real-valued data. Our first contribution is the development and analysis of gradient-based optimization procedures, including stochastic gradient descent and quasi-Newton methods, for learning the parameters of a functional tensor-train (FT) that yields improved accuracy over standard alternating least squares methods. The functional tensor-train uses the tensor-train (TT) representation of low-rank arrays as an ansatz for a class of low-multilinear-rank functions. The FT is represented by a set of matrix-valued functions that contain a set of univariate functions, and the regression task is to learn the parameters of these univariate functions. Our second contribution demonstrates that using nonlinearly parameterized univariate functions, e.g., symmetric kernels with moving centers, within each core can outperform the standard approach of using a linear expansion of basis functions. Our final contributions are new rank adaptation and group-sparsity regularization procedures to minimize overfitting. We use several benchmark problems to demonstrate at least an order of magnitude lower accuracy with gradient-based optimization methods than ALS in the low-sample number regime. We also demonstrate an order of magnitude reduction in accuracy on a test problem resulting from nonlinear parameterizations over linear parameterizations. Finally we compare regression performance with 22 other nonparametric and parametric regression methods on 10 real-world data sets. We achieve top-five accuracy for seven of the data sets and best accuracy for two of the data sets. These rankings are the best amongst parametric models and competetive with the best non-parametric methods.

Keywords. Tensors, Regression, Function Approximation, Alternating Least Squares, Stochastic Gradient Descent

1 Introduction

Assesment of uncertainty in a computational model is essential to increasing the credibility of simulation based knowledge discovery, prediction and design. Sources of model uncertainty must be identified and the effect of these uncertainties on the model output (prediction) quantified. The accuracy to which uncertainty can be quantified is limited by the computational resources available to simulations that solve these governing equations. Many applications require vast amounts of computational effort, thereby limiting the number model evaluations that can be used to interrogate the uncertainty in the system behavior. Consequently a significant portion of methods developed for uncertainty quantification (UQ) in recent years have focused on constructing surrogates of expensive simulation models using only a limited number of model evaluations.

Within the computational science community, both parametric and non-parametric function approximation methods have been extensively used for Uncertainty Quantification (UQ). Non parametric Gaussian process models (GP) [Rasmussen and Williams, 2006, O'Hagan and Kingman, 1978] and parametric Polynomial Chaos Expansions (PCE) [Ghanem and Spanos, 1991, Xiu and Karniadakis, 2002] are arguably the two most popular methods used. Gaussian process regression can be interpreted as a Bayesian method for function approximation, providing a posterior probability distribution over functions.

Disclosure: See acknowledgements
Correspondence author: A.A. Gorodetsky (goroda@umich.edu)
*Department of Aerospace Engineering University of Michigan Ann Arbor, MI, 48109
†This work was performed while the author was at Sandia National Laboratories
‡Optimization and Uncertainty Quantification, Sandia National Laboratories, Albuquerque, NM, 87123

Polynomial chaos expansions represent a response surface as a linear combination of orthonormal multivariate polynomials. The choice of the orthonormal polynomials is related to the distribution of the uncertain model variables. Various approaches have been adopted to compute the coefficients of the PCE basis. Approaches include, pseudo-spectral projection [Conrad and Marzouk, 2013, Constantine et al., 2012], sparse grid interpolation [Ganapathysubramanian and Zabaras, 2007, Nobile et al., 2008], and regression using ℓ_1-minimization [Blatman and Sudret, 2011, Doostan and Owhadi, 2011, Mathelin and Gallivan, 2012]. For a comparison between nonparametric GP methods and parametric PCE methods see, e.g., [Gorodetsky and Marzouk, 2016], and for an attempt to combine the benefits of both approaches see [Schobi et al., 2015].

High-dimensional approximation problems, such as regression, pose challenges for both parametric and nonparametric representation formats. Parametric approaches, for example those using a PCE representation, are limited by their expressivity; and increasing expressivity, for example by increasing the polynomial order, results in the curse of dimensionality for fixed polynomial order.

Nonparametric methods, for example Gaussian process regression, have great expressive capabilities. However, they also encounter the curse of dimensionality since their excess risk grows exponentially with dimension [Györfi et al., 2002].

To counteract these computational burdens for both types of methods, attention has focused on constraining the functional representation to limit the curse of dimensionality. One popular constraint is limiting the model to that of additive separable forms [Hastie and Tibshirani, 1990, Meier et al., 2009, Ravikumar et al., 2008]

$$f(x) = f_1(x_1) + f_2(x_2) + \dots f_d(x_d). \tag{1}$$

One can also use second order interactions, e.g., $f_{12}(x_1, x_2), f_{13}(x_1, x_3), \dots$, to increase expressivity while maintaining tractability [Kandasamy and Yu, 2016]. However, further increasing the number of interaction terms in the ANOVA model [Fisher, 1925]

$$f(x) = \sum_i f_i(x_i) + \sum_{i,j} f_{ij}(x_i, x_j) + \sum_{i,j,k} f_{ijk}(x_i, x_j, x_k) + \cdots$$

will still naturally encounter the curse of dimensionality unless adaptive methods that identify the order of interaction interactively are utilized [Ganapathysubramanian and Zabaras, 2007, Ma and Zabaras, 2010, Foo and Karniadakis, 2010, Jakeman and Roberts, 2013, Jakeman et al., 2015].

In this paper, we propose algorithms to improve regression in a functional representation that takes advantage of *low-rank* structure to mitigate this curse of dimensionality while maintaining high expressivity. Low-rank functional representations are parametric models that enable a wide variety of interactions between variables and can generate high order representations. More specifically, they are continuous analogues of tensor decompositions and exploit *separability*, i.e., that a function can be represented by the sum of products of univariate functions. One example is the canonical polyadic (CP) [Carroll and Chang, 1970] representation consisting of a sum of products of univariate functions $f(x) = \sum_{i=1}^{R} f_1^{(i)}(x_1) \dots f_d^{(i)}(x_d)$, and the number of free parameters of such a representation scales linearly with dimension. Instead of the CP format, we use a continuous analogue of the discrete tensor train (TT) decomposition [Oseledets, 2011] called the functional tensor-train (FT) [Oseledets, 2013, Gorodetsky et al., 2016] to allow for a greater number of interactions between variables.

Low-rank functional tensor decompositions have been used for regression previously. Existing approaches [Doostan et al., 2013, Mathelin, 2014, Chevreuil et al., 2015, Rauhut et al., 2017] implicitely make two simplifying assumptions to facilitate the use of certain algorithms and data structures from the low-rank tensor decomposition literature. Specifically, they assume linear and identical basis expansions for each univariate function of a particular dimension. These approaches convert the problem from one of determining a low-rank *function* to one of representing the coefficients of a tensor-product basis as a low-rank *tensor*.

Following this conversion, many of these techniques use alternating minimization to determine the coefficients of the FT. Alternating minimization, such as alternating least squares, transforms a nonlinear optimization problem for fitting the parameters of each univariate function to data into a linear problem by only considering a single dimension at a time. Existing approaches either use efficient linear algebra routines to solve the linear system at each iteration [Doostan et al., 2013] or sparsity inducing methods such as the LASSO [Mathelin, 2014]. Recentlym iterative thresholding has also been used to find low rank coefficients; however, such an approach has been limited to problems with low-dimensionality [Rauhut et al., 2017].

In this paper we take a different approach: we use gradient-based optimization procedures such as quasi-Newton methods and stochastic gradient descent, and we do not restrict ourselves by the assumptions that lead to the consideration of a tensor of coefficients. Overally, our contributions include:

1. Derivation and computational complexity analysis of a gradient-based fitting procedure that yields more accurate approximations than alternating minimization
2. Usage of both linear and nonlinear parameterizations of univariate function in each mode to facilitate a larger class of functions than is possible when using solely linear representations, and
3. Creation of rank-adaptation and regularization schemes to reduce overfitting.

Our results suggest that gradient-based optimization provides signficant advantages in terms of approximation error, and that these advantages are especially apparent in the case of small amounts of data and large parameter sizes. To our knowledge, no gradient based procedure has been derived or evaluated for the low-rank functional format that we consider. We also show the benefits of nonlinear representations of univariate functions over linear representations. While almost all literature on low-rank functional decomposition thus far has used only linear parameterizations, our results suggest a new area for further research.

Finally, we demonstrate our results on both synthetic functions and on several real-world data sets. We demonstrate an order of magnitude lower accuracy with gradient-based optimization methods than ALS in the low-sample number regime. We also demonstrate an order of magnitude reduction in accuracy on a test problem resulting from nonlinear parameterizations over linear parameterizations. Our real-world data results show that our methodology is competitive with both nonparametric and parametric algorithms. We achieve top-five accuracy for seven of the data sets and best accuracy for two of the data sets. These rankings are the best amongst parametric models and competetive with the best non-parametric methods.

1.1 Related Work

As mentioned above, the functional tensor-train decomposition was proposed as an ansatz for representing functions by Oseledets [2013], and computational routines for compressing a function into FT format have been developed before [Gorodetsky et al., 2016]. In that setting, an approximation of a black-box function is sought to a prescribed accuracy. A sampling procedure and associated algorithm was designed to optimally evaluate the function in order to obtain an FT representation. In this work, we consider the setting of fixed data.

There has also been some recent work on regression in low-rank format [Doostan et al., 2013, Mathelin, 2014, Chevreuil et al., 2015]. These approaches rely on linearity between the parameters of the low-rank format and the output of the function. Utilizing this relationship, they convert the low-rank function approximation to one of low-rank tensor decomposition for the coefficients of a tensor-product basis. In Section 3.2 we show how the representation presented in those works can be obtained as a particular instantiation of the formulation we present here.

In spirit, our approach is also similar to the recent use of the tensor-train decomposition within fully connected neural networks [Novikov et al., 2015]. There, the layers of a neural network are assumed to be a linear transformation of an input vector, and the weight matrix of the transformation is estimated from data. Their contribution is representing the weight matrix in a compressed TT-format. In this work, our low-rank format can be thought of as an alternative to the linear transformation layer of a neural network. Indeed, future work can focus on utilizing our proposed methodology within the layers of a neural network.

2 Background

In this section we establish notation and providing background for regression and low-rank tensor decompositions.

2.1 Notation

Let $\mathbb{R}$ be the set of real numbers and $\mathbb{Z}_+$ be the set of positive integers. Let $n \in \mathbb{Z}_+$, $d \in \mathbb{Z}_+$ and suppose that we are given i.i.d. data $\left(x^{(i)}, y^{(i)}\right)_{i=1}^{n}$ such that each datum is sampled from a distribution μ_{xy} on a

compact space $(x^{(i)}, y^{(i)}) \in \mathcal{X} \times \mathcal{Y} \subset \mathbb{R}^d \times \mathbb{R}$. Let $\mathcal{X} = \mathcal{X}_1 \times \cdots \times \mathcal{X}_d$ be a tensor product space with $\mathcal{X}_k \subset \mathbb{R}$. Let the marginal distribution of $x = (x_1, \ldots, x_d) \in \mathcal{X}$ be the tensor product measure $\mu_x = \mu_{x_1} \times \cdots \times \mu_{x_d}$, and assume that all integrals appearing in this paper are with respect to this measure. For example let $f : \mathcal{X} \to \mathcal{Y}$ and $g : \mathcal{X} \to \mathcal{Y}$, then the inner product is defined as $\langle f, g \rangle = \int f(x)g(x)d\mu(x)$. Similarly, the L_2 norm is defined as $\|f\|_2^2 = \langle f, f \rangle$.

In this paper scalars and scalar-valued functions are denoted by lowercase letters, vectors are denoted by bold lower-case letters such as $\mathbf{x}, \mathbf{y}$; matrices are denoted with upper boldface such as $\mathbf{X}, \mathbf{Y}$; tensors are denoted with upper boldface caligraphic letters such as $\boldsymbol{\mathcal{X}}, \boldsymbol{\mathcal{Y}}$; and matrix-valued functions are denoted by upper caligraphic letters such as $\mathcal{F}, \mathcal{G}$. An ordered sequence of elements of the same set are distinguished by parenthesized superscripts such as $x^{(i)}$ or $y^{(i)}$.

2.2 Supervised learning

The supervised learning tasks seeks a function $f : \mathcal{X} \to \mathcal{Y}$ that minimizes the average of a cost function $g_f : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$

$$J[f] = \int_{\mathcal{X}\times\mathcal{Y}} g_f(x, y)d\mu(xy), \tag{2}$$

For example, the standard least squares objective is specified with $g_f(x, y) = (y - f(x))^2$. In this work, the cost functional cannot be exactly calculated; instead, data $\left(x^{(i)}, y^{(i)}\right)_{i=1}^n$ is used to estimate its value. The cost functional then becomes a sum over the data instead of an integral

$$J\left[f\right] = \frac{1}{n}\sum_{i=1}^{n}\left(y^{(i)} - f\left(x^{(i)}\right)\right)^2, \tag{3}$$

where we have reused the notation $J[f]$, and further references to J will use this definition. To obtain an optimization problem over a finite set of variables, the search space of functions must be restricted. Nonparametric representations generally allow this space to vary depending upon the data, and parameteric representations typically fix the representation. In either case, we denote this space as $\mathscr{F}$ to seek an optimal function $f^* \in \mathscr{F}$ such that

$$f^* = \arg\min_{f\in\mathscr{F}} J[f]. \tag{4}$$

One example of a common function space is the reproducing kernel Hilbert space, and resulting algorithms using this space include Gaussian process regression or radial basis function interpolation. Other examples include linear functions (resulting in linear regression) or polynomial functions. In this work, we consider a function space that incorporate all functions that have a prescribed *rank*, which will be defined in the next section.

When solving the supervised learning problem (4) it is often useful to introduce a regularization term to minimize overfitting. In this paper we will consider the following regularized learning problem

$$f^* = \arg\min_{f\in\mathscr{F}} J[f] + \lambda\Omega[f], \tag{5}$$

where $\lambda \in \mathbb{R}_+$ denotes a scaling factor and $\Omega : \mathscr{F} \to \mathbb{R}_+$ is a functional that penalizes certain functions in $\mathscr{F}$. For example, the function $\Omega(f) = \|f\|_2$ penalizes functions that have large L_2 norms, and such a penalty has been used for a certain type of low-rank functional approximation before [Doostan et al., 2013]. In this work, we impose a group sparsity type regularization that has been shown to improve the prediction performance in other approximation settings [Turlach et al., 2005, Yuan and Lin, 2006]. Such an approach seeks to increase parsimony by reducing the number of non-zero elements in an approximation. In Section 4.1 we describe what this type of constrains means in the context of low-rank functional decompositions.

2.3 Low-rank tensor decompositions

The function space that we use to constrain our supervised learning problem is related to the concept of low-rank decompositions of multiway arrays, or *tensors*. Tensor decompositions are multidimensional

analogues of the matrix SVD and are used to mitigate the curse of dimensionality associated with storing multivariate arrays [Kolda and Bader, 2009]. Consider an array with $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{p_1 \times \cdots \times p_d}$, this array contains an exponentially growing number of elements as the dimension increases. Tensor decompositions can provide a compact representation of the tensor and have found wide spread usage for data compression and learning tasks [Cichocki et al., 2009, Novikov et al., 2015, Yu and Lie, 2016].

In this work, we consider the tensor-train (TT) decomposition [Oseledets, 2011]. Specifically, we use the TT as an ansatz for representating low-rank functions. A TT representation of a tensor $\boldsymbol{\mathcal{A}}$ is defined by a list of 3-way arrays $(\boldsymbol{\mathcal{A}}_k \in \mathbb{R}^{r_{k-1} \times p_k \times r_k})_{k=1}^d$ called TT-cores where $r_0 = r_d = 1$ and $r_k \in \mathbb{Z}_+$ for $k = 2, \ldots, d-1$. The sizes of the cores $(r_k)_{k=0}^d$ are called the *TT-ranks*. In this format a tensor element is obtained according to the following multiplication

$$\boldsymbol{\mathcal{A}}[i_1, i_2, \ldots, i_d] = \boldsymbol{\mathcal{A}}_1[i_1]\boldsymbol{\mathcal{A}}_2[i_2] \cdots \boldsymbol{\mathcal{A}}_d[i_d], \quad 1 < i_k < p_k \text{ for all } k, \tag{6}$$

where $\boldsymbol{\mathcal{A}}_k[i_k] \in \mathbb{R}^{r_{k-1} \times r_k}$ are matrices and the above equation describes a sequence of matrix multiplication.

In this format, storage complexity scales *linearly* with dimension and quadratically with TT-rank. This tensor representation not only mitigates the curse of dimensionality but, unlike the canonical representation, also provides a well posed representation format. In the next section we discuss how it can be used as a framework for function approximation.

3 Low-rank functions

Low-rank formats for functions can be thought of as direct analogues to low-rank tensor decompositions. In this work, we focus on aa TT-like decomposition of functions called the functional tensor-train (FT) [Oseledets, 2013].

$$f(x_1, x_2, \ldots, x_d) = \sum_{i_0=1}^{r_0} \sum_{i_1=1}^{r_1} \cdots \sum_{i_d=1}^{r_d} f_1^{(i_0 i_1)}(x_1) f_2^{(i_1 i_2)}(x_2) \ldots f_d^{(i_{d-1} i_d)}(x_d), \tag{7}$$

where $f_k^{(ij)} : \mathcal{X}_k \to \mathbb{R}$, and $r_0 = r_d = 1$ for single-output functions. A more compact expression is obtained by viewing a function value as a set of products of matrix-valued functions

$$f(x_1, x_2, \ldots, x_d) = \mathcal{F}_1(x_1)\mathcal{F}_2(x_2) \ldots \mathcal{F}_d(x_d), \tag{8}$$

where each matrix-valued function $\mathcal{F}_k : \mathcal{X}_k \to \mathbb{R}^{r_{k-1} \times r_k}$ is called a *core* and can be visualized as an array of the univariate functions

$$\mathcal{F}_k(x_k) = \begin{bmatrix} f_k^{(11)}(x_k) & \cdot & f_k^{(1 r_k)}(x_k) \\ \vdots & \ddots & \vdots \\ f_k^{(r_{k-1} 1)}(x_k) & \cdot & f_k^{(r_{k-1} r_k)}(x_k) \end{bmatrix}. \tag{9}$$

If each univariate function is represented with p parameters and $r_k < r$ for all k, then the storage complexity scales as $\mathcal{O}(dpr^2)$. Comparing this representation with (6) we see a very close resemblence between the TT cores and the FT cores. Indeed they are both matrices when indexed by a discrete index i_k for the TT or a continuous index x_k for the FT. We describe a closer relationship between the TT and the FT in the context of low-rank representations of functions in the next section.

3.1 Parameterizations of low-rank functions

An FT core is comprised of d sets of univariate functions as shown in Figure 1. Each set of univariate functions, contains all of the parameters of the associated univariate functions. As a result the full FT is parameterized through the parameterization of its univariate functions. Let $p_{kij} \in \mathbb{Z}_+$ denote the number of parameters describing $f_k^{(ij)}$. Let $\boldsymbol{\theta}$ denote the vector of parameters of all the univariate functions. Then, there are a total of $\sum_{k=1}^d \sum_{i=1}^{r_{k-1}} \sum_{j=1}^{r_k} p_{kij}$ parameters describing the FT, i.e., $\boldsymbol{\theta} \in \mathbb{R}^{p_t}$.

The parameter vector $\boldsymbol{\theta}$ is indexed by a multi-index $\boldsymbol{\alpha} = (k, i, j, \ell)$ where $k = 1, \ldots, d$ corresponds to an input variable, $i = 1, \ldots, r_{k-1}$ and $j = 1, \ldots, r_k$ correspond to a univariate function within the kth core,

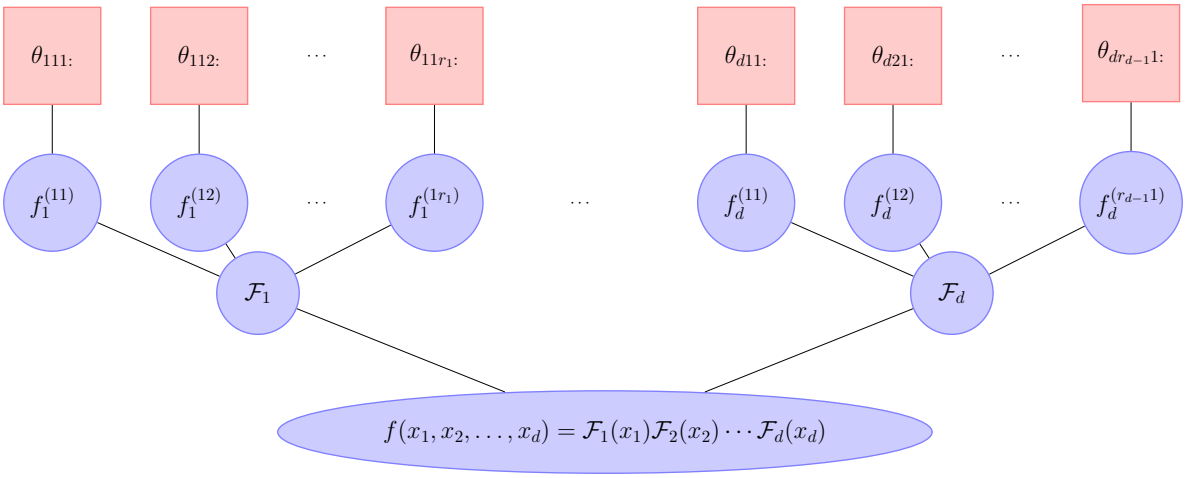


Figure 1: Object-oriented hierarchy for storing low-rank functions. Parameters are represented by red squares and functions are represented in blue.

and $\ell = 1, \ldots, p_{kij}$ corresponds to a specific parameter within that univariate function. In other words, we adopt the convention that $\boldsymbol{\theta}_{\boldsymbol{\alpha}} = \boldsymbol{\theta}_{kij\ell}$ refers to the ℓth parameter of the univariate function in the ith row and jth column of the kth core.

The additional flexibility of the FT representation allows both linear and nonlinear parameterizations of univariate functions. A linear parameterization represents a univariate function as an expansion of basis functions $\left(\phi_{k\ell}^{(ij)} : \mathcal{X}_k \to \mathbb{R}\right)_{\ell=1}^{p_{kij}}$ according to

$$f_k^{(ij)}(x_k; \boldsymbol{\theta}) = \sum_{\ell=1}^{p_{kij}} \theta_{kij\ell} \phi_{k\ell}^{(ij)}(x_k). \tag{10}$$

Linear parameterizations are often convenient within the context of gradient-based optimization methods since the gradient with respect to a parameter is independent of the parameter values. Thus, one only needs to compute and store the evaluation of the basis functions a single time. Nonlinear parameterizations are more flexible and general, but often incur a greater computational cost within optimization. One example of a nonlinear parameterization is that of a Gaussian kernel, which can be written as

$$f_k^{(ij)}(x_k; \boldsymbol{\theta}) = \sum_{\ell=1}^{p_{kij}/2} \theta_{kij\ell} \exp\left(-\frac{1}{\sigma^2}\left(x_k - \theta_{kij(p_{kij}/2+\ell)}\right)^2\right), \tag{11}$$

where $\sigma > 0$ Here, the first $p_{kij}/2$ parameters refer to the coefficients of radial basis functions and that second half of the parameters refer to the centers.

3.2 Low-rank functions vs. low-rank coefficients

The functional tensor train can be used by independently parameterizing the univariate functions of each core, and both linear and nonlinear parameterizations are possible. As described below, the advantage of this representation includes a naturally sparse storage scheme for the cores. Another advantage is the availablitiy of efficient computational algorithms for multilinear algebra that can adapt the representation of each univariate function individually as needed [Gorodetsky et al., 2016, Gorodetsky, 2017] in the spirit of continuous computation pioneered by Chebfun [Platte and Trefethen, 2010].

Although the functional form of the tensor train is very powerful, most of the literature makes two simplifying assumptions [Doostan et al., 2013, Mathelin, 2014, Chevreuil et al., 2015]:

1. a *linear* parameterization of each $f_{k\ell}^{(ij)}$, and
2. an *identitical basis* for the functions within each FT core, i.e., $p_{kij} = p_k$ and $\phi_{k\ell}^{(ij)} = \phi_{k\ell}$ for all $i = 1, \dots r_{k-1}$, $j = 1, \dots, r_k$, and $\ell = 1, \dots, p_k$.

These assumptions transform the problem of learning low-rank functions to the problem of learning low-rank coefficients. This transformation of the problem allows one to facilitate the use of discrete TT algorithms and theory. We will refer to representations using these assumptions as a functional tensor-train with TT coefficients or FT-c, where "c" stands for coefficients.

The FT-c representation stores the coefficients of a tensor-product basis $\phi_{k\ell}$ for all k and ℓ in TT format. Let $\boldsymbol{\mathcal{F}}_k \in \mathbb{R}^{r_{k-1} \times p_k \times r_k}$ be a tensor of the following form

$$\boldsymbol{\mathcal{F}}_k[:, \ell, :] = \begin{bmatrix} \theta_{k11\ell} & \cdot & \theta_{k1r_k\ell} \\ \vdots & \ddots & \vdots \\ \theta_{kr_{k-1}1\ell} & \cdot & \theta_{kr_{k-1}r_k\ell} \end{bmatrix}, \tag{12}$$

for $\ell = 1, \dots, p_k$. Function evaluations can be obtained from the from coefficients stored in TT format by performing the the following summation

$$f(x_1, \dots, x_d) = \sum_{\ell_1=1}^{p_1} \cdots \sum_{\ell_d=1}^{p_d} \boldsymbol{\mathcal{F}}_1[:, \ell_1, :] \cdots \boldsymbol{\mathcal{F}}_d[:, \ell_d, :] \phi_{1\ell_1}(x_1) \cdots \phi_{d\ell_d}(x_d). \tag{13}$$

From (8), (12), and (13) we can see that the relationship between the TT cores $\boldsymbol{\mathcal{F}}_k$ and the FT cores $\mathcal{F}_k$ is

$$\mathcal{F}_k(x_k) = \sum_{\ell=1}^{p_k} \boldsymbol{\mathcal{F}}_k[:, \ell, :] \phi_{k\ell}(x_k), \tag{14}$$

where the basis function multiplies every element of the tensor. In other words, the TT cores represent a TT decomposition of the $p_1 \times p_2 \times \cdots \times p_d$ coefficient tensor of a tensor-product basis.

The two assumptions required for converting the problem from one of low-rank function approximation to low-rank coefficient approximation often result in a computational burden in practice. The burden of the first assumption is clear, some functions are more compressible if nonlinear parameterizations are allowed. The second assumption can also limit compressibility, but also results in a larger computational and storage burden. One example in which this burden becomes obvious is when attempting to represent *additively separable* functions, e.g., Equation (1), in the FT format. As mentioned in the introduction, this representation is common for high-dimensional regression, and an FT can represent this function using cores that fave the following rank 2 structure.

$$f(x_1, x_2, \dots, x_d) = [f_1(x_1)\ 1] \begin{bmatrix} 1 & 0 \\ f_2(x_2) & 1 \end{bmatrix} \cdots \begin{bmatrix} 1 \\ f_d(x_d) \end{bmatrix}.$$

Suppose that each of the univariate functions can be represented with p_k parameters and the constants 0 and 1 can be stored with a single parameter. Then, the storage requirement is $p_k + 3$ parameters for the $d-2$ middle cores and $p_k + 1$ for the outer cores, totaling $(p_{kij} + 3)d - 4$ floating point values. In the TT case, the 0 and 1 terms must be stored with the *same* complexity as the other terms, since the TT is a three-way array. Thus, the total number of parameters becomes $4p_k(d-2)$. Almost four times less storage is required for the FT format than the TT format, in the limit of large p_k and d. Essentially, the TT format does take into account the *sparsity* of the cores. In this case, one can think of the FT format as storing the TT cores in a *sparse* format. This burden is exacerbated if we add interaction terms to Equation 1.

4 Low-rank supervised learning

In this section, we incorporate the FT model as a constraint in the supervised learning task. We discuss issues surrounding optimization algorithms, regularization, and choosing the TT-ranks. In particular we discuss three optimization algorithms for fitting fixed-rank models: batch gradient descent, alternating least squares, and stochastic gradient descent. We also present approaches for minimizing over-fitting. Specifically we discuss a group-sparsity-inducing regularization regularization scheme and hyper-parameter estimation scheme using cross validation and a rank-adapation.

4.1 Low-rank constraints and regularization

The function space $\mathscr{F}$ in Equation (4) constrains the search space. The space constrained by low-rank functions $\mathscr{F}_{\mathbf{r}}$ is defined as follows. Let $\mathbf{r} = [1, r_1, \ldots, r_{d-1}, 1]$ such that $r_i \in \mathbb{Z}_+$ for $i = 1, \ldots, d-1$. Then

$$\mathscr{F}_{\mathbf{r}} = \left\{ f : \mathcal{X} \to \mathbb{R} \;\middle|\; \begin{array}{l} f = \mathcal{F}_1(x_1)\mathcal{F}_2(x_2)\ldots\mathcal{F}_d(x_d) \\ \mathcal{F}_1 : \mathcal{X}_1 \to \mathbb{R}^{1\times r_1} \\ \mathcal{F}_k : \mathcal{X}_k \to \mathbb{R}^{r_{k-1}\times r_k},\ 2 \le k < d \\ \mathcal{F}_d : \mathcal{X}_d \to \mathbb{R}^{r_{d-1}\times 1} \end{array} \right\}$$

denotes the space of functions with FT ranks $\mathbf{r}$. Note that this function space also includes functions with smaller ranks. For example, a function that differs in ranks $\hat{r}_1, \hat{r}_2$ such that $\hat{r}_1 < r_1$ and $\hat{r}_2 < r_2$ can be obtained by setting all univariate functions, aside from the top left $\hat{r}_1 \times \hat{r}_2$ block, to zero; analogous modifications must be made to all other cores.

In this paper we also add a regularization to attempt to minimize the number of nonzero functions in each core. The structure of the FT-cores in Equation (9) admits a natural grouping in terms of the univariate functions. Setting a univariate function to zero inside of a core essentially lowers the number of interactions allowable between univariate functions in neighbouring dimensions, with the overall effect beign a smaller number of sums in the representation of the function in Equation (7). Minimizes the number of nonzero univariate functions not only produces a more parsimonious model, but also minimizes the number of interactions between function variables.

Specifically, we penalize the regression problem using the sum of the norms of the univariate functions

$$\Omega[f] = \sum_{k=1}^{d} \sum_{i=1}^{r_{k-1}} \sum_{j=1}^{r_k} \|f_k^{(ij)}\|^2. \tag{15}$$

Note that this regularization method is different from that first proposed in [Mathelin, 2014] where the FT-c representation was used and the TT-cores of the coefficient tensor themselves were forced to be sparse. In our case we do not look for sparse coefficients, instead our goal is to increase the number of functions that are identically zero. As a surrogate for this goal we follow the standard practice of replacing the non-differential "L_0" norm by the another norm, in our case a sum of the norms of the functions. This replacement also seeks to directly minimize the number of interacting univariate functions and also provides a differentiable objective for optimization.

4.2 Hyper-parameter estimation using cross validation and rank adaptation

A careful selection of the number of parameters p_{kij} in each univariate function, the rank r_k of each core, and the Lagrangian multiplier λ in the regularized learning problem is required to avoid over fitting. For example if a polynomial basis is chosen, setting the degree too high can result in spurious oscillations. Cross validation provides a mechanism to estimate hyper-parameters. In this paper we use 20 fold cross validation to select the hyper-parameters of an FT that minimizes the expected prediction error.

Alternatively, we can design a more effective rank adaptation scheme by combining cross validation with a rounding procedure. The problem with the simplest cross validation option is that a scheme that optimizes separately over each r_k imposes a computational burden, and a scheme that optimizes for a single rank across all cores, i.e., $r_k = r$ does not allow enough flexibility. Instead, we propose to combine FT rounding [Gorodetsky et al., 2016] and cross validation to avoid overfitting.

Rounding is a procedure to reapproximate an FT by one with lower ranks to a given tolerance. The tolerance criterion can be thought of as a regularization term, since it limits the complexity of the representation and will be investigated in furture work. The full rank adaptation scheme is provided by Algorithm 1. In that algorithm $\texttt{cv}(\mathbf{r})$ refers to a function that provides a cross-validation error for an optimization over the space $\mathscr{F}_{\mathbf{r}}$, and the function $\texttt{rounding-rank}(f, \delta)$ provides the ranks of a rounded function f with a particular tolerance δ.

The scheme increases ranks until either the cross-validation error increases *or* until the rounding procedure decreases every rank. The first termination criterion targets overfitting, and the second termination criterion seeks to limit rank growth when data is no longer informative enough to necessitate the increase in expressivity.

Algorithm 1 Rank adaptation

Require: Rounding tolerance δ

$\mathbf{r} = \texttt{ones}(d+1)$
$\epsilon = \texttt{cv}(\mathbf{r})$
while not converged **do**
  for $k = 1, \dots d-1$ **do**
    $r_k \leftarrow r_k + 1$
  end for
  $\mathbf{r} = [1, r_1, \dots, r_{d-1}, 1]$
  $\hat{\epsilon} = \texttt{cv}(\mathbf{r})$
  if $\hat{\epsilon} > \epsilon$ **then**
    $\mathbf{r} = [1, r_1 - 1, \dots, r_{d-1} - 1, 1]$
    break
  end if
  $\epsilon = \hat{\epsilon}$
  $f = \min_{f \in \mathscr{F}_{\mathbf{r}}} J(\boldsymbol{\theta})$
  $[1, \hat{r}_1, \dots, \hat{r}_{d-1}, 1] = \texttt{rounding-rank}(f, \delta)$
  if $\hat{r}_k < r_k$ for all $k = 1, \dots, d-1$ **then**
    break
  else
    $r_k = \hat{r}_k$ for $k = 1, \dots, d-1$
  end if
end while

4.3 Optimization algorithms

In this section, we overview three gradient-based optimization algorithms for solving supervised learning problems in low-rank format: alternating minimization, gradient decent and stochastic gradient decent. The computational complexity of these algorithms in analyzed in Section 5.

4.3.1 Alternating minimization

Alternating minimization methods, mainly alternating least squares, are the main avenue for optimization in low-rank tensor and functional contexts. These routines are typically used within tensor decompositions by sequentially sweeping over dimensions and optimizing over parameters of a single dimension. Such approaches are popular for compressing multiway arrays or for tensor completion problems [Savostyanov and Oseledets, 2011, Grasedyck et al., 2015].

For the case of supervised learning, their usage is straightforward. The idea is to solve a sequence of optimization problems, where the functional space $\mathscr{F}_{\mathbf{r}}$ is further constrained by fixing *all-but-one* FT core, $\mathcal{F}_k$. After optimizing over the kth core, that core is fixed and optimization over the next one is performed. This algorithm is provided by Algorithm 2. This algorithm performs sweeps over all dimensions until convergence is obtained. For details on convergence of this algorithm we refer to, e.g., Uschmajew [2012]. We will provide more details about the implementation and complexity of this algorithm in Section 5.

4.3.2 Batch gradient methods

Gradient descent directly minimizes the cost function $J(\boldsymbol{\theta})$ with a batch gradient (or second-order) based procedure. While this is a standard optimization approach, we will refer to this algorithm as *all-at-once* (AAO) to distinguish it from the alternating minimization. Gradient-based procedures have been shown to be promising in the context of canonical tensor decompositions and gradient descent [Acar et al., 2011] and TT decompositions in the context of iterative thresholding [Rauhut et al., 2017], but have not been explored well for low-rank functional approximation.

Gradient descent updates parameters according to

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \nabla J(\boldsymbol{\theta}),$$

Algorithm 2 Alternating minimization for low-rank supervised learning

Require: Parameterized low-rank function space, $\mathscr{F}_{\mathbf{r}}$; Initial FT cores $\mathcal{F}_k$ for $k = 1, \dots, d$; Data, $\left(x^{(i)}, y^{(i)}\right)_{i=1}^{N}$; Objective function, J; Convergence tolerence, ϵ

$i = 0$
$f^{(i)} = \mathcal{F}_1 \cdots \mathcal{F}_d$
while not converged **do**
  for $k = 1, \dots d$ **do**
    $\mathcal{F}_k = \arg\min_{\hat{\mathcal{F}}_k} J[\mathcal{F}_1 \cdots \mathcal{F}_{k-1} \hat{\mathcal{F}}_k \mathcal{F}_{k+1} \cdots \mathcal{F}_d]$
  end for
  $f^{(i+1)} = \mathcal{F}_1 \cdots \mathcal{F}_d$
  if $\|f^{(i+1)} - f^{(i)}\| \leq \epsilon$ **then**
    break;
  end if
  $i = i + 1$
end while

for $\eta > 0$. More generally, we use a quasi-Newton method to choose a direction $\mathbf{p}$ such that the update becomes

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \mathbf{p}$$

In the examples in Section 6, we use the limited-memory BFGS [Liu and Nocedal, 1989] method available in the C^3 library[1] to perform this update.

One disadvantage of this approach that is often cited is that it involves solving an optimization problem whose size is the number of parameters in the function. However, we note that the number of parameters scales as $\mathcal{O}(dr^2p)$, so for a one hundred dimensional, rank 5 problem with $p_{kij} = p = 5$ we have ≈ 12500 unknowns. This number of unknowns is well within the capabilities of modern hardware and optimization techniques.

4.3.3 Stochastic gradient descent

Stochastic gradient descent (SGD) is often used instead of gradient decent when using large data sets. SGD aims to minimize objective functions of the form

$$J[\boldsymbol{\theta}] = \sum_{i=1}^{n} g_{\boldsymbol{\theta}}(x^{(i)}, y^{(i)}),$$

which includes the least-squares objective (3). The objective function is updated using only one data point (batches can also be used), which is chosen randomly at each iteration

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \nabla g_{\boldsymbol{\theta}}(x^{(i)}, y^{(i)}), \tag{16}$$

Many variations on stochastic gradient descent have been developed. We refer the reader to [Bottou et al., 2016] for more information. These variations often include adaptive strategies for choosing the learning rate η. Such methods have previous been applied in the context of tensors to minimize computation costs when dealing with large scale data [Huang et al., 2015].

We will use the adaptive strategy from ADAM [Diederik and Jimmy, 2014], as implemented in the C^3 library, to demonstrated the effectiveness of SGD in Section 6.

5 Gradient computation and analysis

The evaluation of the gradients of the FT with respect to its parameters is essential for the making gradient-based optimization algorithms tractable. Almost all existing literature for tensor approximation uses alternating minimization strategies because the subproblems are convex and can be solved exactly and efficiently

[1] github.com/goroda/Compressed-Continuous-Computation

with standard linear algebra algorithms, though they have limitied (if any) guarantees for obtaining a good minimizer of the full problem. One of the major contributions of this paper is the derivation and analysis of the computational complexity of computing gradients of the supervised learning objective (4).

The gradient of the least squares objective function with respect to a parameter is

$$\frac{\partial J}{\partial \theta_{\boldsymbol{\alpha}}} = \frac{2}{n}\sum_{i=1}^{n}\left(y^{(i)} - f\left(x^{(i)};\boldsymbol{\theta}\right)\right)\frac{\partial f\left(x^{(i)};\boldsymbol{\theta}\right)}{\partial \theta_{\boldsymbol{\alpha}}}, \tag{17}$$

where $\boldsymbol{\alpha} = (k, i, j, \ell)$ denotes a multi-index for a unique parameter of the kth core, see Section 3.1. Efficient computation of the partial derivative, $\frac{\partial f(x;\boldsymbol{\theta})}{\partial \theta_{\boldsymbol{\alpha}}}$ is essential for making gradient based optimization algorithms tractable. Letting

$$\mathcal{F}_{<k}(x_{<k}) = \mathcal{F}_1(x_1)\cdots\mathcal{F}_{k-1}(x_{k-1}), \quad \text{and} \tag{18}$$
$$\mathcal{F}_{>k}(x_{>k}) = \mathcal{F}_{k+1}(x_{k+1})\cdots\mathcal{F}_d(x_d). \tag{19}$$

denote the products of the cores before and after the kth, respectively, and combining Equations (8), (18), and (19) we obtain the following expression

$$\partial_{\boldsymbol{\alpha}} f(x) \equiv \frac{\partial f(x;\boldsymbol{\theta})}{\partial \theta_{\boldsymbol{\alpha}}} = \mathcal{F}_{<k}(x_{<k})\partial_{\boldsymbol{\alpha}}\mathcal{F}_k(x_k)\mathcal{F}_{>k}(x_{>k}), \tag{20}$$

Thus efficiently computing gradients of the objective function and the FT requires efficient algorithms for evaluation of the left and right product cores $\mathcal{F}_{<k}(x_{<k})$ and $\mathcal{F}_{>k}(x_{>k})$ and the partial derivative $\partial_{\boldsymbol{\alpha}}\mathcal{F}_k(x_k)$.

In the following sections we describe and analyze the gradient of the FT with respect to its parameters. We first describe the partial derivatives of an FT with respect to the parameters of a single core. Then we present an efficient algorithm for computing left and right product cores. Finally, we discuss the computation of gradients of the full FT and the objective function.

5.1 Derivatives of an FT core

In this section we discuss how to obtain the partial derivatives with respect to parameters of a specific core. Without loss of generality , we will make the following assumption to ease the notational burden

Assumption 1. *Each univariate function in each FT-core* (9) *is independently parameterized with p parameters so that the FT core $\mathcal{F}_k$ is parameterized with $r_{k-1}r_k p$ parameters.*

Under this assumption the FT cores have the following structure.

Proposition 1. *Let $\mathbf{G}_{kijl} \in \mathbb{R}^{r_{k-1}\times r}$ denote the partial derivative $\frac{\partial \mathcal{F}_k(z)}{\partial \theta_{\boldsymbol{\alpha}}} = \frac{\partial \mathcal{F}_k(z)}{\partial \theta_{kij\ell}}$ for some $z \in \mathcal{X}_k$. Under Assumption 1, $\mathbf{G}_{kijl}$ is a sparse matrix with the following elements*

$$\mathbf{G}[\alpha,\beta] = \begin{cases} \frac{\partial f_k^{(ij)}(x_k)}{\partial \theta_{kij\ell}} & \text{if } \alpha = i, \beta = j \\ 0 & \text{otherwise} \end{cases} \tag{21}$$

for $\alpha = 1,\ldots,r_{k-1}$, and $\beta = 1,\ldots r_k$.

Now if we let $G(p)$ denote the number of operations required to compute the gradient of a univariate function with respect to *all* of its parameters then the cost of computing $\partial_{\boldsymbol{\alpha}}\mathcal{F}_k(x_k)$ is $\mathcal{O}\left(G(p)r_{k-1}r_k\right)$ operations.

5.2 Evaluating left and right product cores

Computation of the partial derivative of the FT (20) requires the evaluation of the left and right product cores $\mathcal{F}_{<k}(x_{<k})$ and $\mathcal{F}_{>k}(x_{>k})$. A single forward and backward sweep the cores can be used to obtain these values using the following recursion identities

$$\mathcal{F}_{<k+1}(x_{<k+1}) = \mathcal{F}_{<k}(x_{<k})\mathcal{F}_k(x_k) \quad \text{and} \quad \mathcal{F}_{>k-1}(x_{>k-1}) = \mathcal{F}_k(x_k)\mathcal{F}_{>k}(x_{<k}).$$

Algorithm 3 `coregrad-left`: Generate intermediate result for gradient computation

Require: Number of rows, r_{k-1}; Number of columns r_k; Number of parameters per univariate function p; FT core $\mathcal{F}_k$; Left multiplier $\mathbf{a}$

1: **for** $i = 1, \dots r_{k-1}$ **do**
2: **for** $j = 1, \dots r_k$ **do**
3: **for** $\ell = 1, \dots, p$ **do**
4: $\partial f_{kijl} = \mathbf{a}[i] \frac{\partial f_k^{(ij)}(x_k)}{\partial \theta_{kij\ell}}$
5: **end for**
6: **end for**
7: **end for**

Algorithm 4 `coregrad-right`: Update intermediate gradient result with multiplier from the right

Require: Number of rows, r_{k-1}; Number of columns r_k; Number of parameters per univariate function p; Intermediate result ∂f_{kijl}; Right multiplier $\mathbf{c}$

1: **for** $i = 1, \dots r_{k-1}$ **do**
2: **for** $j = 1, \dots r_k$ **do**
3: **for** $\ell = 1, \dots, p$ **do**
4: $\partial f_{kij\ell} = \partial f_{kij\ell} \mathbf{c}[j]$
5: **end for**
6: **end for**
7: **end for**

Thus we can obtain the gradient of the FT with respect to parameters of the kth core using the Algorithms 3 and 4.

These two algorithms consist of a triple nested loop. The innermost loop of `coregrad-left` requires computing the gradient of a univariate function with respect to its parameters, and multipling each element of the gradient by the left multiplier. Thus if $r_k < r$ then $\mathcal{O}(G(p)r^2)$ operations are needed for the gradient and $\mathcal{O}(pr^2)$ products between floating point operations are needed. Finally, the partial derivative with respect to each parameter of the core is stored with a complexity of $\mathcal{O}(pr^2)$ floating point numbers. Each of these partial derivatives is updated in `coregrad-right` for a computational cost of $\mathcal{O}(pr^2)$ and no additional storage. Since each of these functions is called d times, the additional computational cost they incur is $\mathcal{O}(dr^2(G(p) + p))$ and the additional storage complexity they incur is $\mathcal{O}(dpr^2)$.

5.3 Gradient of FT and objective functions

Th entire gradient of the FT can be obtained with a single forward and backward sweep as presented in Algorithm 5. In Algorithm 5 we use $\partial f_{k\boldsymbol{\alpha}_k}$ to denote the partial derivative of f with respect to the parameters of the kth core. Each step of the forward sweep requires: (i) creating an intermediate result for the gradient of the FT with respect to each parameter of the core in line 4 of Algorithm 3, (ii) evaluating and storing the core of the FT in line 6 of Algorithm 5, and (iii) updating the product of the cores through the current step in line 7 of Algorithm 5). The backward sweep involves involves updating the the intermediate gradient result obtained from the forward sweep (line 4 in Algorithm 4), and then updating the product of the cores in line 14 of Algorithm 5.

Proposition 2. *Let $f : \mathcal{X} \to \mathbb{R}$ be a rank $\mathbf{r} = [1\ r_1 \dots r_{d-1}\ 1]$ FT with every univariate function in Equation (9). Assume Assumption 1 and that for $k = 1, \dots, d-1$ we have $r_k < r$ for some $r \in \mathbb{Z}_+$. Let $E(p)$ denote the number of operations required to evaluate a univariate function. Let $G(p)$ denote the number of operations required to compute the gradient of a univariate function with respect to all of its parameters. Then, evaluating $f(x; \boldsymbol{\theta})$ and computing the gradient $\partial_{\boldsymbol{\alpha}} f(x; \boldsymbol{\theta})$ with respect to its parameters requires*

$$\mathcal{O}\left(dr^2\left(G(p) + E(p) + p\right)\right)$$

operations, and storing $\mathcal{O}(dpr^2)$ floating point values.

Algorithm 5 FT evaluation and gradient

Require: Parameters $\boldsymbol{\theta}$; Evaluation location $x = (x_1, x_2, \ldots, x_d) \in \mathcal{X}$
1: {Generate intermediate results during forward sweep}
2: $\mathbf{a} = \mathcal{F}_1(x_1)$
3: $\partial f_{1\boldsymbol{\alpha}_1} = \texttt{coregrad-left}(r_0, r_1, p, \mathcal{F}_1, \mathbf{a})$
4: **for** $k = 2, \ldots, d$ **do**
5: $\quad \partial f_{k\boldsymbol{\alpha}_k} = \texttt{coregrad-left}(r_{k-1}, r_k, p, \mathcal{F}_k, \mathbf{a})$
6: $\quad \mathbf{F}_k = \mathcal{F}_k(x_k)$
7: $\quad \mathbf{a} \leftarrow \mathbf{a}\mathbf{F}_k$
8: **end for**
9: $f(x; \boldsymbol{\theta}) = \mathbf{a}[1]$ {Evaluation of the FT}
10: {Update intermediate results during backward sweep}
11: $\mathbf{c} = \mathbf{F}_d$
12: **for** $k = d-1, \ldots, 2$ **do**
13: $\quad \partial f_{k\boldsymbol{\alpha}_k} = \texttt{coregrad-right}(r_{k-1}, r_k, p, \partial f_{k\boldsymbol{\alpha}_k}, \mathbf{c})$
14: $\quad \mathbf{c} \leftarrow \mathbf{F}_k\mathbf{c}$
15: **end for**
16: $\partial f_{1\boldsymbol{\alpha}_1} = \texttt{coregrad-right}(r_0, r_1, p, f_{1\boldsymbol{\alpha}_1}, \mathbf{c})$

Proof To demonstrate this result, we can calculate the number of evaluations and storage size required for each step of Algorithm 5. First note that Lines 2 and 6 involve the evaluation of each FT core. Since each core has at most r^2 univariate functions these evaluations require $E(p)r^2$ operations and the ability to store r^2 floating point numbers. Since this evaluation has to happend for each of the d cores the computational complexity of this step is $dE(p)r^2$. The storage space can be reused and only requires storing two vectors of size $r \times 1$ ($\mathbf{a}$) and an $r \times r$ matrix ($\mathbf{F}_k$) for a total storage complexity of $\mathcal{O}(r^2 + r) = \mathcal{O}(r^2)$ floating point numbers. Next we note that a product between a $1 \times r$ vector and an $r \times r$ matrix in lines 7 and 14 needs to occur $2d$ times and therefore requires $\mathcal{O}(dr^2)$ operations. Thus apart from the calls to `coregrad-left` and `coregrad-right` we have a computational complexity of $\mathcal{O}(dE(p)r^2)$ and a storage complexity of $\mathcal{O}(r^2)$.

Combining these costs with the cost of the coregrad algorthims presented in Section 5.2 obtain the stated result. □

The values $G(p)$ and $E(p)$ are dependent upon the types of parameterizations of univariate functions. Consider two examples: one linear and one nonlinear. For both parameterizations we consider kernel basis functions; however, for the nonlinear parameterization we will consider the centers of each kernel as free parameters. The linear parameterization is given by Equation (10) with $\phi_{k\ell}^{(ij)}(x_k) = \exp\left(-\frac{1}{\sigma^2}\left(x_k - c_{k\ell}\right)^2\right)$, where $\sigma \in \mathbb{R}_+$ and $c_{k\ell} \in \mathcal{X}_k$ are the centers of the kernel such that each univariate function of the kth dimension is represented with as a sum of kernels with the same locations. If the evaluation of the exponential takes a constant number of operations with respect to x_k then we have $E(p) = \mathcal{O}(p)$ and $G(p) = \mathcal{O}(p)$ because the the gradient of the univariate function with respect to its ℓth parameter is

$$\frac{\partial f_k^{(ij)}(x_k; \boldsymbol{\theta})}{\partial \theta_{kij\ell}} = \exp\left(-\frac{1}{\sigma^2}\left(x_k - c_{k\ell}\right)^2\right). \tag{22}$$

This gradient is independent of any other parameters. In practice this means that it can be precomputed at each location x_k and recalled at runtime. In such a case the storage increases to $\mathcal{O}(ndpr^2)$ numbers if each univariate function in each core has a different parameterization. If the univariate functions of each core share the same parameterization then the additional storage cost is $\mathcal{O}(npd)$. In either case the online cost becomes a simple lookup, i.e., $G(p) = \mathcal{O}(1)$.

The nonlinear parameterization provided in Equation (11) is different because we are free to optimize over the centers. In this case the gradient of each univariate function becomes The gradient with respect to the second half of the parameters now depends on the parameter value

$$\frac{\partial f_k^{(ij)}(x_k; \boldsymbol{\theta})}{\partial \theta_{kij\ell}} = \begin{cases} \theta_{kij\ell} \exp\left(-\frac{1}{\sigma^2}\left(x_k - \theta_{kij(p_{kij}/2+\ell)}\right)^2\right) & \text{for } \ell = 1, \ldots, p/2 \\ \frac{2}{\sigma^2}\theta_{kij(\ell-p/2)}\left(x_k - \theta_{kij\ell}\right)\exp\left(-\frac{1}{\sigma^2}\left(x_k - \theta_{kij\ell}\right)^2\right) & \text{for } \ell = p/2+1, \ldots, p \end{cases}$$

Table 1: Computational complexity of all-at-once optimization.

Parameterization type	Offline cost/storage	Eval. and grad. (S)	Full solution
Shared/linear	$\mathcal{O}(ndG(p))$ / $\mathcal{O}(ndp)$	$\mathcal{O}(ndr^2\left(E(p)+p\right))$	$\mathcal{O}(N_{\text{opt,AAO}}S)$
General/linear	$\mathcal{O}(ndr^2G(p))$ / $\mathcal{O}(ndr^2p)$	$\mathcal{O}(ndr^2\left(E(p)+p\right))$	$\mathcal{O}(N_{\text{opt,AAO}}S)$
nonlinear	N/A	$\mathcal{O}\left(ndr^2\left(G(p)+E(p)+p\right)\right)$	$\mathcal{O}(N_{\text{opt,AAO}}S)$

Table 2: Computational complexity of stochastic gradient descent with ADAM.

Parameterization type	Offline cost/storage	Eval. and grad. per sample (S)	Full solution
Shared/linear	$\mathcal{O}(ndG(p))$ / $\mathcal{O}(ndp)$	$\mathcal{O}(dr^2\left(E(p)+p\right))$	$N_{\text{epoch}}nS$
General/linear	$\mathcal{O}(ndr^2G(p))$ / $\mathcal{O}(ndr^2p)$	$\mathcal{O}(dr^2\left(E(p)+p\right))$	$N_{\text{epoch}}nS$
nonlinear	N/A	$\mathcal{O}\left(dr^2\left(G(p)+E(p)+p\right)\right)$	$N_{\text{epoch}}nS$

The parameter dependence of the gradient increases the complexity of optimization algorithms since precomputation of the gradients cannot be performed. In this case we have $E(p)=\mathcal{O}(p)$ and $G(p)=\mathcal{O}(p)$.

5.4 Summary of computational complexity

The complexity of evaluating the gradient of the least squares objective function

$$\frac{\partial J}{\partial \theta_{\boldsymbol{\alpha}}}=\frac{2}{n}\sum_{i=1}^{n}\left(y^{(i)}-f\left(x^{(i)};\boldsymbol{\theta}\right)\right)\frac{\partial f\left(x^{(i)};\boldsymbol{\theta}\right)}{\partial \theta_{\boldsymbol{\alpha}}}$$

is dominated by the cost of this derivative is evaluating the gradient of f at n points. Consequently, the total complexity is n times that provided by Proposition 2, for a total cost of $\mathcal{O}(ndr^2(G(p)+E(p)))$ operations. The storage cost need not increase because one can evaluate the sum by sequentially iterating through each sample.

Now that we have described the computational complexity of the gradient computation, we summarize the computational complexity of the proposed optimization algorithms. Table 1 shows the computational complexity of the all-at-once optimization scheme when using a low-memory BFGS solver. Three parameterizations are considered a linear parameterization with identical parameterizations of each univariate function in a particular core, a more general parameterization with varying linear parmaeterizations within each core and, and the most general case of nonlinear parameterization of each function. We see that using linear parameterizations allows us to precompute certain gradients before running the optimizer. This precomputation reduces the online cost of optimization. The computational complexity of the full solution is dominated by the number of evaluations and gradients of the objective function, and we denote this number as $N_{\text{opt,AAO}}$.

The computational cost of stochastic gradient descent is given in Table 2. In this case the cost per epoch (once through all of the training points) is the same as a single gradient evaluation in the all-at-once approach. The total cost of such a scheme is dominated by how many samples are used during the optimization procedures. In the associated table, we represent this number as the number of times one requires iterating through all of the samples $N_{\text{epoch}}n$. A fully online algorithm would not have any associated offline cost and its complexity would be equivalent to the nonlinear parameterization case.

Finally, the alternating optimization scheme is of a slightly different nature. In the case of linear parameterization, each sub-optimization problem is quadratic and can be posed as solving a linear system by setting the right hand side of Equation (17) to zero. This linear system has n equations and pr^2 unknowns and its solution, using a direct method, has an asymptotic complexity of $\mathcal{O}\left(np^2r^4\right)$. We also need to introduce a new constant called N_{sweep} that represents how many sweeps through all of the dimensions are required. We see that in the most general case, this algorithm is N_{sweep} times more expensive than all-at-once. However, this number is a bit desceptive since each sub problem has only pr^2 unknowns and therefore we can assume that $N_{\text{opt,ALS}}<N_{\text{opt,AAO}}$. In Table 3 we summarize the costs of this algorithm.

The summaries above were provided for the pure least squares regression problem. If we consider the regularization term of Equation (15) then an additional step must be performed. The gradient of the

Table 3: Computational complexity of alternating (non)-linear least squares.

Param type	Offline cost/storage	Solution of sub-problem (S)	Full solution
Shared/linear	$\mathcal{O}(ndG(p))$ / $\mathcal{O}(ndp)$	$\mathcal{O}\left(np^2r^4\right)$	$\mathcal{O}\left(N_{\text{sweeps}}dS\right)$
General/linear	$\mathcal{O}\left(ndr^2G(p)\right)$ / $\mathcal{O}\left(ndr^2p\right)$	$\mathcal{O}\left(np^2r^4\right)$	$\mathcal{O}\left(N_{\text{sweeps}}dS\right)$
nonlinear	N/A	$\mathcal{O}\left(N_{\text{opt,ALS}}nr^2\left(G(p)+E(p)+p\right)\right)$	$\mathcal{O}\left(N_{\text{sweeps}}dS\right)$

functional $\Omega[f]$ requires computing the gradient of each of the summands, which can be written as

$$\frac{\partial \int_{\mathcal{X}_k}\left[f_k^{(ij)}\left(x_k;\theta_{kr_{k-1}r_k1},\ldots,\theta_{kr_{k-1}r_kp}\right)\right]^2 dx_k}{\partial\theta_{kr_{k-1}r_k\ell}} =$$
$$2\int_{\mathcal{X}_k}\frac{\partial f_k^{(ij)}\left(x_k;\theta_{kr_{k-1}r_k1},\ldots,\theta_{kr_{k-1}r_kp}\right)}{\partial\theta_{kr_{k-1}r_k\ell}} f_k^{(ij)}(x_k;\theta_{kr_{k-1}r_k1},\ldots,\theta_{kr_{k-1}r_kp})dx_k,$$

for $\ell = 1,\ldots,p$. In other words, to compute the gradient of every element in the sum in Equation (15), one computes the inner product between the original function and the function representing the partial derivative of this function. In the case of linear parameterizations described above, the partial derivative is simply a scalar and only the integral of the univariate function needs to be computed. Alternatively the full inner product must be evaluated; however we note that the evaluation of this integral is often analytical or available in closed form based upon the type of function. For example the cost is $\mathcal{O}(p)$ for orthonormal basis functions and $\mathcal{O}(p^2)$ more generally, to obtain the gradient with respect to every parameter of a univariate function.

6 Experiments

In this section, we provide experimental validation of our approach. Our validation is performed on a synthetic function, approximation benchmark problems, and several real-world data sets. The synthetic examples are used to show the convergence of our approximations with increasing number of samples using the various optimization approaches. We also include comparisons between both the nonlinear parameterized FT and the linearly parameterized FT-c representations. Furthermore, we show the effectiveness of our rank adaptation approach. The real-world data sets indicate the viability of FT-based regression for a wide variety of application areas and indicates pervasiveness of low-rank structure.

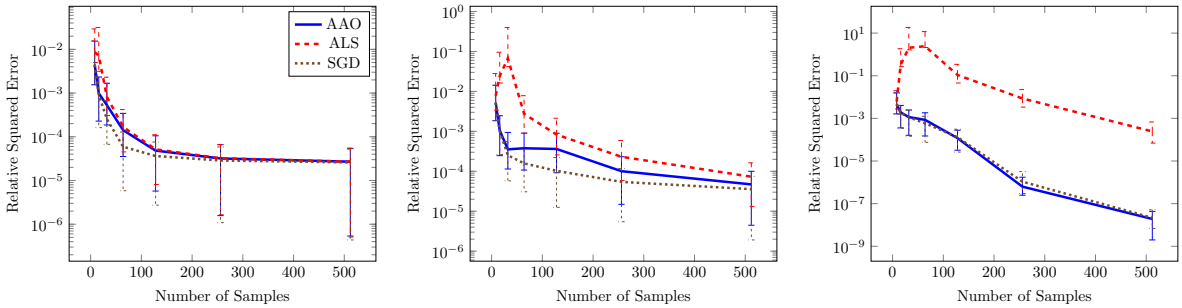
6.1 Comparison of optimization strategies

We first compare the convergence, with the number of samples, of three optimization algorithms. We use three synthetic test cases with known rank and parameterization. The first two functions are from a commonly used benchmark database [Surjanovic and Bingham, 2017]. The third function is a FT-rank 2 function that is commonly used to demonstrate the performance of low-rank algorithms.

The first function is six dimensional and called the OTL circuit function. It is given by

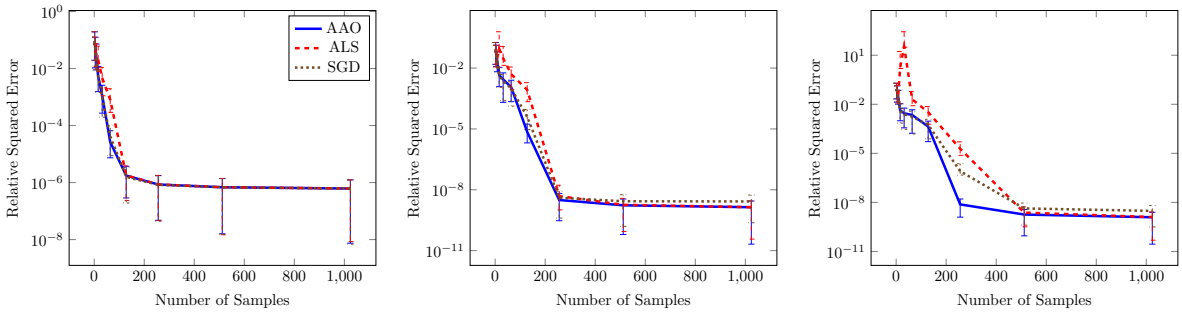
$$f(R_{b1},R_{b2},R_f,R_{c1},R_{c2},\beta) = \frac{(V_{b1}+0.74)\beta(R_{c2}+9)}{\beta(R_{c2}+9)+R_f} + \frac{11.35R_f}{\beta(R_{c2}+9)+R_f} + \frac{0.74R_f\beta(R_{c2}+9)}{(\beta(R_{c2}+9)+R_f)R_{c1}}, \quad (23)$$

with $V_{b1} = \frac{12R_{b2}}{R_{b1}+R_{b2}}$ and variable bounds $R_{b1}\in[50,150]$, $R_{b2}\in[25,70]$, $R_f\in[0.5,3]$, $R_{c1}\in[1.2,2.5]$, $R_{c2}\in[0.25,1.2]$, and $\beta\in[50,300]$. This function has a decaying spectrum due to the complicated sum of variables in the denominators, and it provides and an important test problem for our proposed rank adaptation scheme. Before exploring rank-adaptation first we explore the effectiveness of the three optimization algorithms in the context of fixed rank r and number of univariate parameters p. Specifically we compute the relative squared error, using 10000 validation samples, for increasing number of training samples. We use a stopping criterion of 10^{-13} for the difference between objective values for the gradient based techniques and for the difference between functions of successive sweeps for ALS. Though we have found that the results for these



(a)$r = 2, p = 3$ (b)$r = 4, p = 3$ (c)$r = 4, p = 9$

Figure 2: Median, 25th, and 75th quantiles of relative error over 100 realizations of training samples for the OTL Circuit (23).



(a)$p = 3$ (b)$p = 5$ (c)$p = 6$

Figure 3: Median, 25th, and 75th quantiles of relative error over 100 realizations of training samples for the Wing Weight function (24).

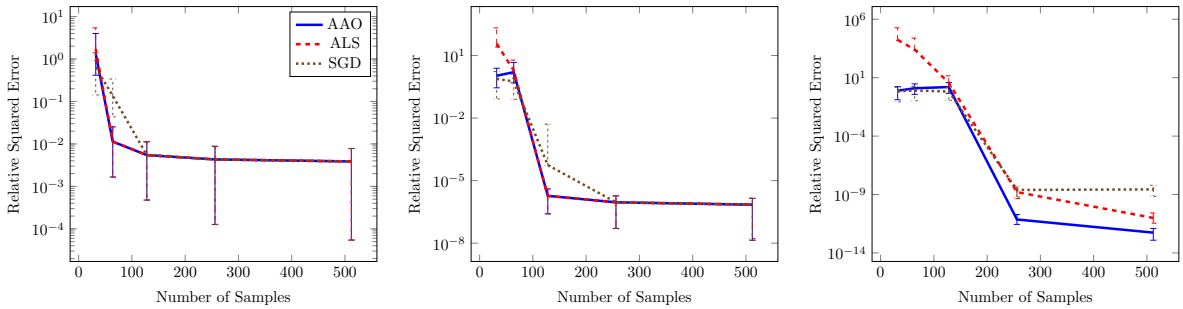
functions are not too sensitive to this tolerance. We solve 100 realizations of the optimization problem for each sample size and report the median, 25th, and 75th quantiles. The univariate function are parameterized using Legendre polynomials because we are using a domain with uniform measure.

Figure 2 demonstrates that stochastic gradient descent and the all-at-once approach tend to outperform alternating least squares. This performance benefit is greater in regions of small sample sizes and large number of unknowns (large r and p). In the case of the largest number of unknowns in Figure 2c our gradient based methods obtain an error that is several orders of magnitude lower error than the error obtained by ALS.

Next we consider two cases in which the rank is known. The wing weight function is ten dimensional and given by

$$f(S_w, W_{fw}, A, \Lambda, q, \lambda, t_c, N_z, W_{dg}, W_p) = \\ 0.036 S_w^{0.758} W_{fw}^{0.0035} \left(\frac{A}{\cos^2(\Lambda)}\right)^{0.6} q^{0.006} \lambda^{0.04} \left(\frac{100 t_c}{\cos(\Lambda)}\right)^{-0.3} (N_z W_{dg})^{0.49} + S_w W_p, \tag{24}$$

with variable bounds $S_w \in [150, 200]$, $W_{fw} \in [220, 300]$, $A \in [6, 10]$, $\Lambda \in [-10, 10]$, $q \in [16, 45]$, $\lambda \in [0.5, 1]$, $t_c \in [0.08, 0.18]$, $N_z \in [2.5, 6]$, $W_{dg} \in [1700, 2500]$, $W_p \in [0.025, 0.08]$. Using the variable ordering above the rank of this function is $r = 2$. The results in Figure 3 indicate the same qualitative performance of the three optimizaiton methods. However, the difference in this case is that the SGD is not significantly better than all-at-once (AAO) approach for low sample sizes. In the third panel we see that SGD levels off around a relative squared error of 10^{-9}. For such small errors we have found it difficult to converge the SGD to smaller errors because of the tuning parameters involved in ADAM. In particular, the final error tolerance becomes sensitive to the choice of initial learning rate.



(a)$p = 3$ (b)$p = 5$ (c)$p = 9$

Figure 4: Median, 25th, and 75th quantiles of relative error over 100 realizations of training samples for the Sine of sums (25).

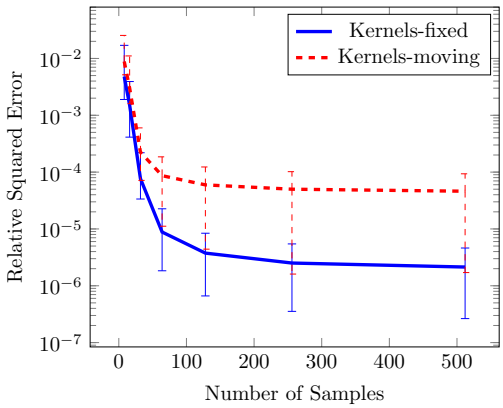


Figure 5: Comparison of parameterization with eight kernels with fixed and uniformly distributed centers and four kernels with with optimized centers with rank $r = 4$. Median, 25th, and 75th quantiles are obtained over 100 realizations of training data.

The final test function is six dimensional and commonly used for testing tensor approximation because it has a TT-rank of 2 [Oseledets and Tyrtyshnikov, 2010], this function is a Sine of sums

$$f(x) = \sin\left(\sum_{i=1}^{6} x_i\right), \quad x_i \in [-1, 1], \quad i = 1, \ldots, 6. \tag{25}$$

In Figure 4 we also see that the gradient based approaches are more effective in the case of small number of data sets, but that all achieve essentially the same minimums as the number of samples increase.

6.2 Linear vs nonlinear approximation

Next we compare the FT and FT-c representations with different basis functions. Specifically, using the OTL function (23), we compare kernels at fixed locations and with kernels at optimized locations. For the linear approximation we use 8 kernels with fixed centers, and for the nonlinear approximation we use 4 kernels with moving centers (for a total of $p = 8$ for both approximation types). The results of this study are shown in Figure 5. Using AAO optimization we see that, for this problem, the nonlinear parameterization of kernels with moving centers provides a more effective representation. Specifically, we achieve an order of magnitude reduction in error when using the nonlinear moving-center representation.

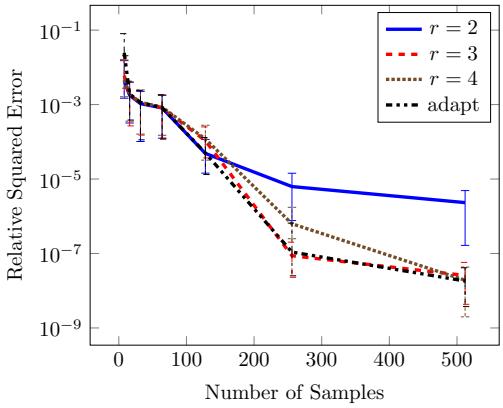


Figure 6: Rank adaptation performance, polynomial order is fixed to eight. Median, 25th, and 75th quantiles are obtained over 100 realizations of training data. The adaptive scheme follows the lowest-error-curve for each of the sample size.

6.3 Rank adaptation

Since the OTL circuit function (23) does not have finite rank, it is a good candidate for exploration of our rank adaptation scheme. With this goal, we compare rank adaptation with three approximations with varying fixed-rank, while fixing the polynomial order to eight. The results, shown in Figure 6, indicate the desired behavior of the rank adaptation scheme, which follows the line of the best fixed-rank approximation. In particular for low sample sizes the rank-2 approximation is the best fixed-rank approximation, and the adaptive scheme shows approximately the same error. Once the sample size increases, the higher rank approximations converge more quickly and the adaptive scheme is able to follow the best approximation.

6.4 Real-data performance

In this section we compare regression in the FT format on real-world data sets and with other regression algorithms. In particular, we use 10 real-world data sets made available by Kandasamy and Yu [2016] for which a set of 22 algorithms, 19 nonparametric and 3 parametric, were compared. The three parametric algorithms included a ridge regularized regression algorithm with linear basis functions, and two sparse regularized algorithms LASSO and LAR. These data sets are sourced from a variety of sources and preprocessed to normalize the inputs and outputs to zero mean and a standard deviation of one. To allow estimation of prediction error, the authors randomly split each data set in half to generate a training sample set and a separate validation set. The mean squared error

$$MSE = \frac{1}{N} \sum_{i=1}^{n_{\text{validation}}} \left(\hat{f}(x^{(i)}) - y^{(i)} \right)^2,$$

is calculated only over the validation set.

In addition to comparing the FT to the other algorithms, we also compare with a robust and effective parameteric approximation scheme based on cross-validated LASSO. This scheme is able to use higher-order polynomials and we have found it to perform better than the results reported in the paper. Let Φ be a Vandermonde-like matrix whose entries $\Phi_{ij} = \phi_j(x_i)$ are the j-th basis function evaluated at the i-th point, then LASSO finds the basis coefficients that minimize

$$\|\Phi\boldsymbol{\theta}\|_2^2 + \lambda_{\text{LASSO}}\|\boldsymbol{\theta}\|_1 \tag{26}$$

We use least angle regression Tibshirani [1996] to solve the LASSO problem and use 10-fold cross validation to choose the regularization parameter λ_{LASSO}. We also use cross validation to simulatenously choose the degree of the total-degree polynomial basis. Only degree-one and degree-two polynomial spaces were considered because the size of the Vandermonde matrix in linear-parameteric representations grows exponentially with

Table 4: Mean squared error on validation data for a subset of regression algorithms and data sets from Kandasamy and Yu [2016]. Only the algorithms which scored the best in at least one data set and the CV-based LASSO are shown. The FT is in the last column and highlighted in blue. Bolded numbers indicate the best performance for each data set and underlined numbers indicate a top five performance out of all 22 algorithms.

Dataset (d, n)	SALSA	nSVR	RT	GBRT	MARS	FT	CVLASSO
Housing (12,256)	**0.26241**	0.38600	1.06015	0.42951	0.42379	0.32798	0.35218
Galaxy (20,2000)	**0.00014**	0.15798	0.02293	0.01405	0.00163	0.00056	0.00249
Skillcraft (18,1700)	0.54695	0.66311	1.08047	0.57273	0.54595	**0.54434**	0.56879
CCPP (59,2000)	0.06782	0.09449	1.04527	**0.06181**	0.08189	0.06631	0.06466
Speech (21,520)	0.02246	0.06994	0.05430	0.03515	**0.01647**	0.02684	0.03131
Music (90,1000)	0.62512	**0.59399**	1.45983	0.66652	0.88779	0.72094	0.64485
Telemonit (19,1000)	0.03473	0.05246	**0.01375**	0.04371	0.02400	0.04040	0.06487
Propulsion (15,200)	0.00881	0.00910	0.02341	0.00061	0.01290	**0.00009**	0.02660
Airfoil (40,750)	0.51756	0.55118	0.45249	**0.34461**	0.54552	0.46920	0.46444
Forestfires (10,211)	0.35301	0.43142	0.41531	**0.26162**	0.33891	0.39465	0.44175

dimension. Note that these examples highlight the fact that we can use expressive basis functions in high high dimensions by exploiting low-rank structure. In other words, low-rank functional decompositions enable more expressive parameterized approximation forms.

For the FT, we use a kernel basis and perform 20-fold cross validation to choose the number of kernels $p \in \{3, 6, 9\}$, the rank $r \in \{1, 2\}$, the kernel width parameters $w \in \{1, 2, 4, 6, 8\}$, and the regularization term $\lambda \in \{10^{-3}, 10^{-7}\}$. The kernel width is chosen similarly to [Kandasamy and Yu, 2016] where we have $\sigma = wn^{1/5}\hat{\sigma}$, and $\hat{\sigma}$ denotes the standard deviation of the input data. Because we do not know the input domains for this data, therefore we position the kernels according to the empirical marginal distribution the data sets along each dimension. Specifically, we position the kernels at uniform quantiles of the data between the 10th and 90th quantiles. Finally, we use the AAO optimization setup, and report the mean squared errors on the testing data in Table 4. The FT is the best model for 2 data sets and in top five for seven data sets, it is the only parametric model that scored the best on at least one data set. The FT also outperforms the four other parametric models that use sparse regression or ridge regression.

Because these data sets come from a wide variety of application areas, these results indicate that low-rank structure exists and is pervasive in a wide variety of regimes.

6.5 Application: modeling a propulsion plant on a naval vessel

In this section we consider a simulation of a gas turbine propulsion engine mounted on a naval Frigate as detailed in Coraddu et al. [2014], and for which simulation data is made openly available through the UCI Machine Learning Repository [Lichman, 2013]. The goal of the UQ problem is to predict the degradation of the gas turbine based on certain parameters of its operation. According to Coraddu et al. [2014], the model for the propulsion system is made of three components: the engine, the transmission gear, and the propulsor. The model is described by a set of nonlinear differential equations.

This simulation has sixteen parameters as detailed in Table 5. The output that we attempt to predict using these simulation inputs is the gas turbine degradation coefficient that describes the gas flow rate reduction factor over service hours.

We use the data provided through the UCI repository to compare our proposed low-rank regression methodology to commonly used sparse regression algorithms. This data consists of 11934 instances of parameters and outputs. We also noticed that as part of this data the GT compressor inlet temperature and the GT compresser inlet air pressure did not vary, but we still included these variables in the regression problem to check if our approaches are robust to such cases.

We use the CV-based LASSO scheme described above using a total order basis of up-to 4th order polynomials and we use a rank adaptive low-rank regression scheme using the AAO approach. We also cross validate for up to 4th order polynomials using 5-fold cross validation, and we limit the BFGS algorithm to 500 iterations. We perform regression for 20 realizations of training data. We consider training sample sizes of $29, 59, 119$, and 238 samples, and we validate on the remaining data.

Table 5: Regression inputs to propulsion simulator from Coraddu et al. [2014]. HP denotes high-pressure and GT denotes gas turbine.

Variable	Units
Lever position	(·)
Ship speed	knot
Gas turbine shaft torque	kN m
Gas turbine rate of revolutions	r/min
Gas generator rate of revolutions	r/min
Starboard propeller torque	kN
Port propeller torque	kN
HP turbine exit temperature	C
GT compressor inlet air temperature	C
GT compressor outlet air temperature	C
HP turbine exit pressure	bar
GT compressor inlet air pressure	bar
GT compressor outlet air pressure	bar
GT exhaust gas pressure	bar
Turbine injection Control	%
Fuel flow	kg / s

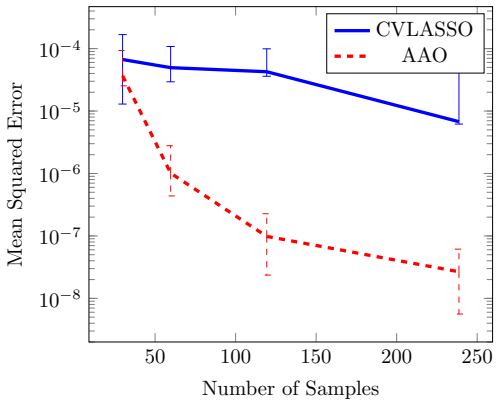


Figure 7: Comparison of rank adaptive low-rank regression with AAO optimization and CV LASSO on simulated data from the gas turbine of a naval vessel propulsion plant from Coraddu et al. [2014].

Figure 7 demonstrate that we are able to achieve several orders of magnitude reduction in MSE with the low-rank approach as compare to the sparse regression approach for this problem. These results suggest that high frequency interactions exist between the input parameters that are not captured using a total order polynomial expansion. Indeed a full tensor product basis is needed, and that the coefficients of this tensor product basis are low rank. Performing sparse regression with the full tensor product basis of 4th order polynomials would have required solving for $5^{16} = \mathcal{O}(10^{11})$, or approximately 152 billion unknowns.

7 Conclusions

We have derived and analyzed the computational complexity of gradient based optimization for regression in a low-rank functional tensor format, the functional tensor-train (FT). Our analysis is valid for both the common FT-c variant, where a tensor-train of coefficients of a tensor product basis is used to represent the function, and for the more general case where the FT is represented using a set of (non)linearly parameterized univariate functions. Furthermore, we have proposed and demonstrated the effectiveness of both a rank-adaptation scheme to prevent overfitting and nonlinear parameterizations of univariate functions to increase

expressivity.

Our results indicate that full gradient-based approaches have significant accuracy advantages over the standard practice of alternating least squares optimization. The accuracy of full gradient based schemes is especially improved in the context of low sample sizes with respect to the number of free parameters. These empirical findings hold true for both batch gradient optimization methods such as BFGS and stochastic gradient descent methods such as ADAM. Furthermore, we have shown that low-rank approximation itself is a promising model for general function approximation and machine learning. In particular, tests on 10 data sets from various application areas indicate that the FT is competetitive, and sometimes better than, 22 other commonly used algorithms.

There exists many directions for future research. The first direction is improving the performance for noisy and/or small data through more effective regularization techniques. Here, we have incorporated a basic group sparsity regularization term, but this area of research is actively being developed and can be expanded into the multilinear context we consider here. Another direction for research is reducing the need to cross validate over the number of parameters in each basis through more adaptive techniques that modify the number of parameters on the fly.

8 Acknowledgements

This work was supported by the DARPA EQUIPS project and by the John von Neumann Postdoctoral Fellowship offered through the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, Applied Mathematics program. Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC., a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA-0003525.

References

C. E. Rasmussen and Christopher K. I. Williams. *Gaussian processes for machine learning*, volume 1. MIT press Cambridge, 2006.

A. O'Hagan and J. F. C. Kingman. Curve fitting and optimal design for prediction. *Journal of the Royal Statistical Society. Series B (Methodological)*, 40(1):1–42, 1978. ISSN 00359246. URL `http://www.jstor.org/stable/2984861`.

R.G. Ghanem and P.D. Spanos. *Stochastic Finite Elements: A Spectral Approach*. Springer-Verlag New York, Inc., New York, NY, USA, 1991.

Dongbin Xiu and George Em Karniadakis. The wiener–askey polynomial chaos for stochastic differential equations. *SIAM journal on scientific computing*, 24(2):619–644, 2002.

Patrick R. Conrad and Youssef M. Marzouk. Adaptive smolyak pseudospectral approximations. *SIAM J. Scientific Computing*, 35(6), 2013.

P.G. Constantine, M.S. Eldred, and E.T. Phipps. Sparse pseudospectral approximation method. *Computer Methods in Applied Mechanics and Engineering*, 229232(0):1–12, 2012. URL `http://www.sciencedirect.com/science/article/pii/S0045782512000953`.

B. Ganapathysubramanian and N. Zabaras. Sparse grid collocation schemes for stochastic natural convection problems. *Journal of Computational Physics*, 225(1):652–685, 2007.

F. Nobile, R. Tempone, and C.G. Webster. An anisotropic sparse grid stochastic collocation method for partial differential equations with random input data. *SIAM Journal on Numerical Analysis*, 46(5):2411–2442, 2008. URL `http://link.aip.org/link/?SNA/46/2411/1`.

Graud Blatman and Bruno Sudret. Adaptive sparse polynomial chaos expansion based on least angle regression. *Journal of Computational Physics*, 230(6):2345 – 2367, 2011. URL http://www.sciencedirect.com/science/article/pii/S0021999110006856.

A . Doostan and H. Owhadi. A non-adapted sparse approximation of PDEs with stochastic inputs. *Journal of Computational Physics*, 230(8):3015 – 3034, 2011. URL http://www.sciencedirect.com/science/article/pii/S0021999111000106.

L Mathelin and KA Gallivan. A compressed sensing approach for partial differential equations with random input data. *Commun. Comput. Phys.*, 12:919–954, 2012.

Alex Gorodetsky and Youssef Marzouk. Mercer kernels and integrated variance experimental design: Connections between gaussian process regression and polynomial approximation. *SIAM/ASA Journal on Uncertainty Quantification*, 4(1):796–828, 2016.

Roland Schobi, Bruno Sudret, and Joe Wiart. Polynomial-chaos-based kriging. *International Journal for Uncertainty Quantification*, 5(2), 2015.

L. Györfi, M. Kohler, A. Krzyzak, and H. Walk. *A distribution-free theory of nonparametric regression*. Springer-Verlag, 2002.

T. J. Hastie and R. J. Tibshirani. *Generalized additive models*. Chapman and Hall, 1990.

L. Meier, S. Van de Geer, and P. Bühlmann. High-dimensional additive modeling. *The Annals of Statistics*, 37(6B):3779–3821, 2009.

P. Ravikumar, H. Liu, J. D. Lafferty, and L. Wasserman. SpAM: Sparse Additive Models. In J. C. Platt, D. Koller, Y. Singer, and S. T. Roweis, editors, *Advances in Neural Information Processing Systems 20*, pages 1201–1208. Curran Associates, Inc., 2008.

K. Kandasamy and Y. Yu. Additive approximations in high dimensional nonparametric regression via the SALSA. In *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, 2016.

R. A. Fisher. *Statistical methods for research workers*. Genesis Publishing Pvt Ltd, 1925.

X. Ma and N. Zabaras. An adaptive high-dimensional stochastic model representation technique for the solution of stochastic partial differential equations. *Journal of Computational Physics*, 229(10):3884–3915, May 2010. ISSN 00219991. doi: 10.1016/j.jcp.2010.01.033. URL http://dx.doi.org/10.1016/j.jcp.2010.01.033.

J. Foo and G.E. Karniadakis. Multi-element probabilistic collocation method in high dimensions. *J. Comput. Phys.*, 229(5):1536–1557, 2010. ISSN 0021-9991. doi: http://dx.doi.org/10.1016/j.jcp.2009.10.043.

J.D. Jakeman and S.G. Roberts. Local and dimension adaptive stochastic collocation for uncertainty quantification. In Jochen Garcke and Michael Griebel, editors, *Sparse Grids and Applications*, volume 88 of *Lecture Notes in Computational Science and Engineering*, pages 181–203. Springer Berlin Heidelberg, 2013. ISBN 978-3-642-31702-6. doi: 10.1007/978-3-642-31703-3_9. URL http://dx.doi.org/10.1007/978-3-642-31703-3_9.

J.D. Jakeman, M.S. Eldred, and K. Sargsyan. Enhancing ℓ_1-minimization estimates of polynomial chaos expansions using basis selection. *Journal of Computational Physics*, 289(0):18 – 34, 2015. ISSN 0021-9991. doi: http://dx.doi.org/10.1016/j.jcp.2015.02.025. URL http://www.sciencedirect.com/science/article/pii/S0021999115000959.

J. D. Carroll and J. Chang. Analysis of individual differences in multidimensional scaling via an N-way generalization of "Eckart-Young" decomposition. *Psychometrika*, 35(3):283–319, 1970.

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

I. V. Oseledets. Constructive representation of functions in low-rank tensor formats. *Constructive Approximation*, 37(1):1–18, 2013.

A. A. Gorodetsky, Sertac Karaman, and Y. M. Marzouk. Function-train: A continuous analogue of the tensor-train decomposition. *arXiv preprint arXiv:1510.09088v2*, 2016.

A. Doostan, A. Validi, and G. Iaccarino. Non-intrusive low-rank separated approximation of high-dimensional stochastic models. *Computer Methods in Applied Mechanics and Engineering*, 263:42–55, 2013.

L. Mathelin. Quantification of uncertainty from high-dimensional scattered data via polynomial approximation. *International Journal for Uncertainty Quantification*, 4(3), 2014.

M. Chevreuil, R. Lebrun, A. Nouy, and P. Rai. A least-squares method for sparse low rank approximation of multivariate functions. *SIAM/ASA Journal on Uncertainty Quantification*, 3(1):897–921, 2015.

Holger Rauhut, Reinhold Schneider, and Željka Stojanac. Low rank tensor recovery via iterative hard thresholding. *Linear Algebra and its Applications*, 523:220–262, 2017.

A. Novikov, D. Podoprikhin, A. Osokin, and D. P. Vetrov. Tensorizing neural networks. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 442–450. Curran Associates, Inc., 2015.

Berwin A Turlach, William N Venables, and Stephen J Wright. Simultaneous variable selection. *Technometrics*, 47(3):349–363, 2005. doi: 10.1198/004017005000000139. URL `http://dx.doi.org/10.1198/004017005000000139`.

Ming Yuan and Yi Lin. Model selection and estimation in regression with grouped variables. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 68(1):49–67, 2006. ISSN 1467-9868. doi: 10.1111/j.1467-9868.2005.00532.x. URL `http://dx.doi.org/10.1111/j.1467-9868.2005.00532.x`.

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

A. Cichocki, R. Zdunek, A. H. Phan, and S. Amari. Nonnegative matrix and tensor factorizations: applications to exploratory multi-way data analysis and blind source separation, 2009.

R. Yu and Y. Lie. Learning from multiway data: Simple and efficient tensor regression. In *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, pages 373–381, 2016.

Alex Arkady Gorodetsky. *Continuous low-rank tensor decompositions, with applications to stochastic optimal control and data assimilation*. PhD thesis, Massachusetts Institute of Technology, 2017.

Rodrigo B Platte and Lloyd N Trefethen. Chebfun: a new kind of numerical computing. *Progress in industrial mathematics at ECMI 2008*, pages 69–87, 2010.

D. Savostyanov and I. V. Oseledets. Fast adaptive interpolation of multi-dimensional arrays in tensor train format. In *Multidimensional (nD) Systems (nDs), 2011 7th International Workshop on*, pages 1–8. IEEE, 2011.

L. Grasedyck, M. Kluge, and S. Krämer. Variants of alternating least squares tensor completion in the tensor train format. *SIAM Journal on Scientific Computing*, 37(5):A2424–A2450, 2015.

André Uschmajew. Local convergence of the alternating least squares algorithm for canonical tensor approximation. *SIAM Journal on Matrix Analysis and Applications*, 33(2):639–652, 2012.

Evrim Acar, Daniel M Dunlavy, and Tamara G Kolda. A scalable optimization approach for fitting canonical tensor decompositions. *Journal of Chemometrics*, 25(2):67–86, 2011.

Dong C. Liu and Jorge Nocedal. On the limited memory bfgs method for large scale optimization. *Mathematical Programming*, 45(1):503–528, Aug 1989. ISSN 1436-4646. doi: 10.1007/BF01589116. URL `https://doi.org/10.1007/BF01589116`.

Léon Bottou, Frank E Curtis, and Jorge Nocedal. Optimization methods for large-scale machine learning. *arXiv preprint arXiv:1606.04838*, 2016.

Furong Huang, UN Niranjan, Mohammad Umar Hakeem, and Animashree Anandkumar. Online tensor methods for learning latent variable models. *Journal of Machine Learning Research*, 16:2797–2835, 2015.

P. K. Diederik and B. Jimmy. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

S. Surjanovic and D. Bingham. Virtual library of simulation experiments: Test functions and datasets. Retrieved November 8, 2017, from `http://www.sfu.ca/~ssurjano`, 2017.

I. V. Oseledets and E. Tyrtyshnikov. TT-cross approximation for multidimensional arrays. *Linear Algebra and its Applications*, 432(1):70–88, 2010.

R. Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 267–288, 1996.

Andrea Coraddu, Luca Oneto, Alessandro Ghio, Stefano Savio, Davide Anguita, and Massimo Figari. Machine learning approaches for improving condition?based maintenance of naval propulsion plants. *Journal of Engineering for the Maritime Environment*, –(–):–, 2014.

M. Lichman. UCI machine learning repository, 2013. URL `http://archive.ics.uci.edu/ml`.