Polynomial Networks and Factorization Machines: New Insights and Efficient Training Algorithms

Mathieu Blondel MATHIEU.BLONDEL@LAB.NTT.CO.JP
Masakazu Ishihata ISHIHATA.MASAKAZU@LAB.NTT.CO.JP
Akinori Fujino FUJINO.AKINORI@LAB.NTT.CO.JP
Naonori Ueda UEDA.NAONORI@LAB.NTT.CO.JP
NTT Communication Science Laboratories, 2-4 Hikaridai Seika-cho Soraku-gun, Kyoto 619-0237, Japan

Abstract

Polynomial networks and factorization machines are two recently-proposed models that can efficiently use feature interactions in classification and regression tasks. In this paper, we revisit both models from a unified perspective. Based on this new view, we study the properties of both models and propose new efficient training algorithms. Key to our approach is to cast parameter learning as a low-rank symmetric tensor estimation problem, which we solve by multi-convex optimization. We demonstrate our approach on regression and recommender system tasks.

1. Introduction

Interactions between features play an important role in many classification and regression tasks. One of the simplest approach to leverage such interactions consists in *explicitly* augmenting feature vectors with products of features (monomials), as in polynomial regression. Although fast linear model solvers can be used (Chang et al., 2010; Sonnenburg & Franc, 2010), an obvious drawback of this kind of approach is that the number of parameters to estimate scales as $O(d^m)$, where d is the number of features and m is the order of interactions considered. As a result, it is usually limited to second or third-order interactions.

Another popular approach consists in using a polynomial kernel so as to *implicitly* map the data via the kernel trick. The main advantage of this approach is that the number of parameters to estimate in the model is actually independent of d and m. However, the cost of storing and evaluating the model is now proportional to the number of training instances. This is sometimes called the curse of kernelization (Wang et al., 2010). Common ways to address the issue include the Nyström method (Williams & Seeger, 2001), random features (Kar & Karnick, 2012) and sketching (Pham & Pagh, 2013; Avron et al., 2014).

In this paper, in order to leverage feature interactions in possibly very high-dimensional data, we consider models which predict the output $y \in \mathbb{R}$ associated with an input vector $\boldsymbol{x} \in \mathbb{R}^d$ by

$$\hat{y}_{\mathcal{K}}(\boldsymbol{x}; \boldsymbol{\lambda}, \boldsymbol{P}) \coloneqq \sum_{s=1}^{k} \lambda_s \mathcal{K}(\boldsymbol{p}_s, \boldsymbol{x}), \tag{1}$$

where $\boldsymbol{\lambda} = [\lambda_1, \ldots, \lambda_k]^{\mathrm{T}} \in \mathbb{R}^k$, $\boldsymbol{P} = [\boldsymbol{p}_1, \ldots, \boldsymbol{p}_k] \in \mathbb{R}^{d \times k}$, $\mathcal{K}$ is a kernel and k is a hyper-parameter. More specifically, we focus on two specific choices of $\mathcal{K}$ which allow us to use feature interactions: the homogeneous polynomial and the ANOVA kernels. Our contributions are as follows. We show (Section 3) that choosing one kernel or the other allows us to recover **polynomial networks** (PNs) (Livni et al., 2014) and, surprisingly, **factorization machines** (FMs) (Rendle, 2010; 2012). Based on this new view, we show important properties of PNs and FMs. Notably, we show for the first time that the objective function of arbitrary-order FMs is multi-convex (Section 4). Unfortunately, the objective function of PNs is not multi-convex. To remedy this problem, we propose a *lifted* approach, based on casting parameter estimation as a low-rank tensor estimation problem (Section 5.1). Combined with a symmetrization trick, this approach leads to a multi-convex problem, for *both* PNs and FMs (Section 5.2). We demonstrate our approach on regression and recommender system tasks.

Notation. We denote vectors, matrices and tensors using lower-case, upper-case and calligraphic bold, e.g., $\boldsymbol{w}$, $\boldsymbol{W}$ and $\boldsymbol{\mathcal{W}}$. We denote the set of $\overbrace{d \times \cdots \times d}^{m \text{ times}}$ real tensors by $\mathbb{R}^{d^m}$ and the set of symmetric real tensors by $\mathbb{S}^{d^m}$. We use $\langle \cdot, \cdot \rangle$ to denote vector, matrix and tensor inner product. Given $\boldsymbol{x}$, we define a symmetric rank-one tensor by

Proceedings of the 33^{rd} International Conference on Machine Learning, New York, NY, USA, 2016. JMLR: W&CP volume 48.

$\boldsymbol{x}^{\otimes m} \coloneqq \boldsymbol{x} \otimes \cdots \otimes \boldsymbol{x} \in \mathbb{S}^{d^m}$, where $(\boldsymbol{x}^{\otimes m})_{j_1,j_2,\ldots,j_m} = x_{j_1} x_{j_2} \ldots x_{j_m}$. We use $[d]$ to denote the set $\{1, \ldots, d\}$.

2. Related work

2.1. Polynomial networks

Polynomial networks (PNs) (Livni et al., 2014) of degree $m = 2$ predict the output $y \in \mathbb{R}$ associated with $\boldsymbol{x} \in \mathbb{R}^d$ by

$$\hat{y}_{\text{PN}}(\boldsymbol{x}; \boldsymbol{w}, \boldsymbol{\lambda}, \boldsymbol{P}) \coloneqq \langle \boldsymbol{w}, \boldsymbol{x} \rangle + \langle \sigma(\boldsymbol{P}^{\mathrm{T}} \boldsymbol{x}), \boldsymbol{\lambda} \rangle, \qquad (2)$$

where $\boldsymbol{w} \in \mathbb{R}^d$, $\boldsymbol{P} \in \mathbb{R}^{d \times k}$, $\boldsymbol{\lambda} \in \mathbb{R}^k$ and $\sigma(u) \coloneqq u^2$ is evaluated element-wise. Intuitively, the right-hand term can be interpreted as a feedforward neural network with one hidden layer of k units and with activation function $\sigma(u)$. Livni et al. (2014) also extend (2) to the case $m = 3$ and show theoretically that PNs can approximate feedforward networks with sigmoidal activation. A similar model was independently shown to perform well on dependency parsing (Chen & Manning, 2014). Unfortunately, the objective function of PNs is non-convex. In Section 5, we derive a *multi-convex* objective based on low-rank symmetric tensor estimation, suitable for training arbitrary-order PNs.

2.2. Factorization machines

One of the simplest way to leverage feature interactions is polynomial regression (PR). For example, for second-order interactions, in this approach, we compute predictions by

$$\hat{y}_{\text{PR}}(\boldsymbol{x}; \boldsymbol{w}, \boldsymbol{W}) \coloneqq \langle \boldsymbol{w}, \boldsymbol{x} \rangle + \sum_{j' > j} \boldsymbol{W}_{j,j'} x_j x_{j'},$$

where $\boldsymbol{w} \in \mathbb{R}^d$ and $\boldsymbol{W} \in \mathbb{R}^{d^2}$. Obviously, model size in PR does not scale well w.r.t. d. The main idea of (second-order) factorization machines (FMs) (Rendle, 2010; 2012) is to replace $\boldsymbol{W}$ with a factorized matrix $\boldsymbol{P}\boldsymbol{P}^{\mathrm{T}}$:

$$\hat{y}_{\text{FM}}(\boldsymbol{x}; \boldsymbol{w}, \boldsymbol{P}) \coloneqq \langle \boldsymbol{w}, \boldsymbol{x} \rangle + \sum_{j' > j} (\boldsymbol{P}\boldsymbol{P}^{\mathrm{T}})_{jj'} x_j x_{j'},$$

where $\boldsymbol{P} \in \mathbb{R}^{d \times k}$. FMs have been increasingly popular for efficiently modeling feature interactions in high-dimensional data, see (Rendle, 2012) and references therein. In Section 4, we show for the first time that the objective function of arbitrary-order FMs is *multi-convex*.

3. Polynomial and ANOVA kernels

In this section, we show that the prediction functions used by polynomial networks and factorization machines can be written using (1) for a specific choice of kernel.

The **polynomial kernel** is a popular kernel for using combinations of features. The kernel is defined as

$$\mathcal{P}^m_\gamma(\boldsymbol{p}, \boldsymbol{x}) \coloneqq (\gamma + \langle \boldsymbol{p}, \boldsymbol{x} \rangle)^m,$$

where $m \in \mathbb{N}$ is the degree and $\gamma > 0$ is a hyper-parameter. We define the **homogeneous polynomial kernel** by

$$\mathcal{H}^m(\boldsymbol{p}, \boldsymbol{x}) \coloneqq \mathcal{P}^m_0(\boldsymbol{p}, \boldsymbol{x}) = \langle \boldsymbol{p}, \boldsymbol{x} \rangle^m.$$

Let $\boldsymbol{p} = [p_1, \ldots, p_d]^{\mathrm{T}}$ and $\boldsymbol{x} = [x_1, \ldots, x_d]^{\mathrm{T}}$. Then,

$$\mathcal{H}^m(\boldsymbol{p}, \boldsymbol{x}) = \sum_{j_1=1}^{d} \cdots \sum_{j_m=1}^{d} p_{j_1} x_{j_1} \ldots p_{j_m} x_{j_m}.$$

We thus see that $\mathcal{H}^m$ uses *all* monomials of degree m (i.e., all combinations of features *with* replacement).

A much lesser known kernel is the **ANOVA kernel** (Stitson et al., 1997; Vapnik, 1998). Following (Shawe-Taylor & Cristianini, 2004, Section 9.2), the ANOVA kernel of degree m, where $2 \leq m \leq d$, can be defined as

$$\mathcal{A}^m(\boldsymbol{p}, \boldsymbol{x}) \coloneqq \sum_{j_m > \cdots > j_1} p_{j_1} x_{j_1} \ldots p_{j_m} x_{j_m}. \qquad (3)$$

As a result, $\mathcal{A}^m$ uses only monomials composed of *distinct* features (i.e., feature combinations *without* replacement). For later convenience, we also define $\mathcal{A}^0(\boldsymbol{p}, \boldsymbol{x}) \coloneqq 1$ and $\mathcal{A}^1(\boldsymbol{p}, \boldsymbol{x}) \coloneqq \langle \boldsymbol{p}, \boldsymbol{x} \rangle$.

With $\mathcal{H}^m$ and $\mathcal{A}^m$ defined, we are now in position to state the following lemma.

Lemma 1 *Expressing PNs and FMs using kernels*

Let $\hat{y}_{\mathcal{K}}(\boldsymbol{x}; \boldsymbol{\lambda}, \boldsymbol{P})$ be defined as in (1). Then,

$$\hat{y}_{PN}(\boldsymbol{x}; \boldsymbol{w}, \boldsymbol{\lambda}, \boldsymbol{P}) = \langle \boldsymbol{w}, \boldsymbol{x} \rangle + \hat{y}_{\mathcal{H}^2}(\boldsymbol{x}; \boldsymbol{\lambda}, \boldsymbol{P})$$
$$\hat{y}_{FM}(\boldsymbol{x}; \boldsymbol{w}, \boldsymbol{P}) = \langle \boldsymbol{w}, \boldsymbol{x} \rangle + \hat{y}_{\mathcal{A}^2}(\boldsymbol{x}; \boldsymbol{1}, \boldsymbol{P}).$$

The relation easily extends to higher orders. This new view allows us to state results that will be very useful in the next sections. The first one is that $\mathcal{H}^m$ and $\mathcal{A}^m$ are homogeneous functions, i.e., they satisfy

$$\lambda^m \mathcal{K}(\boldsymbol{p}, \boldsymbol{x}) = \mathcal{K}(\lambda \boldsymbol{p}, \boldsymbol{x}) \quad \forall \lambda \in \mathbb{R}, \forall m \in \mathbb{N}_+.$$

Another key property of $\mathcal{A}^m(\boldsymbol{p}, \boldsymbol{x})$ is multi-linearity.[1]

Lemma 2 *Multi-linearity of $\mathcal{A}^m(\boldsymbol{p}, \boldsymbol{x})$ w.r.t. $p_1, \ldots, p_d$*

Let $\boldsymbol{p}, \boldsymbol{x} \in \mathbb{R}^d$, $j \in [d]$ and $1 \leq m \leq d$. Then,

$$\mathcal{A}^m(\boldsymbol{p}, \boldsymbol{x}) = \mathcal{A}^m(\boldsymbol{p}_{\neg j}, \boldsymbol{x}_{\neg j}) + \; p_j x_j \; \mathcal{A}^{m-1}(\boldsymbol{p}_{\neg j}, \boldsymbol{x}_{\neg j})$$

where $\boldsymbol{p}_{\neg j}$ denotes the $(d-1)$-dimensional vector with p_j removed and similarly for $\boldsymbol{x}_{\neg j}$.

That is, everything else kept fixed, $\mathcal{A}^m(\boldsymbol{p}, \boldsymbol{x})$ is an affine function of p_j, $\forall j \in [d]$. Proof is given in Appendix B.1.

[1] A function $f(\theta_1, \ldots, \theta_k)$ is called multi-linear (resp. multi-convex) if it is linear (resp. convex) w.r.t. $\theta_1, \ldots, \theta_k$ separately.

Assuming $\boldsymbol{p}$ is dense and $\boldsymbol{x}$ sparse, the cost of naively computing $\mathcal{A}^m(\boldsymbol{p}, \boldsymbol{x})$ by (3) is $O(n_z(\boldsymbol{x})^m)$, where $n_z(\boldsymbol{x})$ is the number of non-zero features in $\boldsymbol{x}$. To address this issue, we will make use of the following lemma for computing $\mathcal{A}^m$ in nearly $O(mn_z(\boldsymbol{x}))$ time when $m \in \{2, 3\}$.

Lemma 3 *Efficient computation of ANOVA kernel*

$$\mathcal{A}^2(\boldsymbol{p}, \boldsymbol{x}) = \frac{1}{2} \left[\mathcal{H}^2(\boldsymbol{p}, \boldsymbol{x}) - \mathcal{D}^2(\boldsymbol{p}, \boldsymbol{x})\right]$$

$$\mathcal{A}^3(\boldsymbol{p}, \boldsymbol{x}) = \frac{1}{6} \left[\mathcal{H}^3(\boldsymbol{p}, \boldsymbol{x}) - 3\mathcal{D}^{2,1}(\boldsymbol{p}, \boldsymbol{x}) + 2\mathcal{D}^3(\boldsymbol{p}, \boldsymbol{x})\right]$$

where we defined $\mathcal{D}^m(\boldsymbol{p}, \boldsymbol{x}) \coloneqq \sum_{j=1}^d (p_j x_j)^m$ *and* $\mathcal{D}^{m,n}(\boldsymbol{p}, \boldsymbol{x}) \coloneqq \mathcal{D}^m(\boldsymbol{p}, \boldsymbol{x})\mathcal{D}^n(\boldsymbol{p}, \boldsymbol{x})$.

See Appendix B.2 for a derivation.

4. Direct approach

Let us denote the training set by $\boldsymbol{X} = [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n] \in \mathbb{R}^{d \times n}$ and $\boldsymbol{y} = [y_1, \ldots, y_n]^{\mathrm{T}} \in \mathbb{R}^n$. The most natural approach to learn models of the form (1) is to directly choose $\boldsymbol{\lambda}$ and $\boldsymbol{P}$ so as to minimize some error function

$$D_{\mathcal{K}}(\boldsymbol{\lambda}, \boldsymbol{P}) \coloneqq \sum_{i=1}^n \ell\left(y_i, \hat{y}_{\mathcal{K}}(\boldsymbol{x}_i; \boldsymbol{\lambda}, \boldsymbol{P})\right), \tag{4}$$

where $\ell(y_i, \hat{y}_i)$ is a convex loss function. Note that (4) is a convex objective w.r.t. $\boldsymbol{\lambda}$ regardless of $\mathcal{K}$. However, it is in general non-convex w.r.t. $\boldsymbol{P}$. Fortunately, when $\mathcal{K} = \mathcal{A}^m$, we can show that (4) is multi-convex.

Theorem 1 *Multi-convexity of* (4) *when* $\mathcal{K} = \mathcal{A}^m$

$D_{\mathcal{A}^m}$ *is convex in* $\boldsymbol{\lambda}$ *and in each row of* $\boldsymbol{P}$ *separately.*

Proof is given in Appendix B.3. As a corollary, the objective function of FMs of arbitrary order is thus multi-convex. Theorem 1 suggests that we can minimize (4) efficiently when $\mathcal{K} = \mathcal{A}^m$ by solving a succession of convex problems w.r.t. $\boldsymbol{\lambda}$ and the rows of $\boldsymbol{P}$. We next show that when m is odd, we can just fix $\boldsymbol{\lambda} = \mathbf{1}$ without loss of generality.

Lemma 4 *When is it useful to fit* $\boldsymbol{\lambda}$*?*

Let $\mathcal{K} = \mathcal{H}^m$ *or* $\mathcal{A}^m$*. Then*

$$\min_{\boldsymbol{\lambda} \in \mathbb{R}^k, \boldsymbol{P} \in \mathbb{R}^{d \times k}} D_{\mathcal{K}}(\boldsymbol{\lambda}, \boldsymbol{P}) \leq \min_{\boldsymbol{P} \in \mathbb{R}^{d \times k}} D_{\mathcal{K}}(\mathbf{1}, \boldsymbol{P}) \quad \textit{if } m \textit{ is even}$$

$$\min_{\boldsymbol{\lambda} \in \mathbb{R}^k, \boldsymbol{P} \in \mathbb{R}^{d \times k}} D_{\mathcal{K}}(\boldsymbol{\lambda}, \boldsymbol{P}) = \min_{\boldsymbol{P} \in \mathbb{R}^{d \times k}} D_{\mathcal{K}}(\mathbf{1}, \boldsymbol{P}) \quad \textit{if } m \textit{ is odd.}$$

The result stems from the fact that $\mathcal{H}^m$ and $\mathcal{A}^m$ are homogeneous functions. If we define $\boldsymbol{v} \coloneqq \operatorname{sign}(\lambda) \sqrt[m]{|\lambda|}\boldsymbol{p}$, then we obtain $\lambda\mathcal{H}^m(\boldsymbol{p}, \boldsymbol{x}) = \mathcal{H}^m(\boldsymbol{v}, \boldsymbol{x})\ \forall \lambda$ if m is odd, and similarly for $\mathcal{A}^m$. That is, λ can be absorbed into $\boldsymbol{v}$ without loss of generality. When m is even, $\lambda < 0$ cannot be absorbed unless we allow complex numbers. Because FMs fix $\boldsymbol{\lambda} = \mathbf{1}$, Lemma 4 shows that the class of functions that FMs can represent is possibly smaller than our framework.

5. Lifted approach

5.1. Conversion to low-rank tensor estimation problem

If we set $\mathcal{K} = \mathcal{H}^m$ in (4), the resulting optimization problem is *neither* convex *nor* multi-convex w.r.t. $\boldsymbol{P}$. In (Blondel et al., 2015), for $m = 2$, it was proposed to cast parameter estimation as a low-rank symmetric matrix estimation problem. A similar idea was used in the context of phase retrieval in (Candès et al., 2013). Inspired by these works, we propose to convert the problem of estimating $\boldsymbol{\lambda}$ and $\boldsymbol{P}$ to that of estimating a low-rank *symmetric* tensor $\boldsymbol{\mathcal{W}} \in \mathbb{S}^{d^m}$. Combined with a symmetrization trick, this approach leads to an objective that is multi-convex, for *both* $\mathcal{K} = \mathcal{A}^m$ and $\mathcal{K} = \mathcal{H}^m$ (Section 5.2).

We begin by rewriting the kernel definitions using rank-one tensors. For $\mathcal{H}^m(\boldsymbol{p}, \boldsymbol{x})$, it is easy to see that

$$\mathcal{H}^m(\boldsymbol{p}, \boldsymbol{x}) = \langle \boldsymbol{p}^{\otimes m}, \boldsymbol{x}^{\otimes m} \rangle. \tag{5}$$

For $\mathcal{A}^m(\boldsymbol{p}, \boldsymbol{x})$, we need to ignore irrelevant monomials. For convenience, we introduce the following notation:

$$\langle \boldsymbol{\mathcal{W}}, \boldsymbol{\mathcal{X}} \rangle_{>} \coloneqq \sum_{j_m > \cdots > j_1} \boldsymbol{\mathcal{W}}_{j_1, \ldots, j_m} \boldsymbol{\mathcal{X}}_{j_1, \ldots, j_m} \quad \forall\, \boldsymbol{\mathcal{W}}, \boldsymbol{\mathcal{X}} \in \mathbb{S}^{d^m}.$$

We can now concisely rewrite the ANOVA kernel as

$$\mathcal{A}^m(\boldsymbol{p}, \boldsymbol{x}) = \langle \boldsymbol{p}^{\otimes m}, \boldsymbol{x}^{\otimes m} \rangle_{>}. \tag{6}$$

Our key insight is described in the following lemma.

Lemma 5 *Link between tensors and kernel expansions*

Let $\boldsymbol{\mathcal{W}} \in \mathbb{S}^{d^m}$ *have a symmetric outer product decomposition (Comon et al., 2008)*

$$\boldsymbol{\mathcal{W}} = \sum_{s=1}^k \lambda_s \boldsymbol{p}_s^{\otimes m}. \tag{7}$$

Let $\boldsymbol{\lambda} = [\lambda_1, \ldots, \lambda_k]^{\mathrm{T}}$ *and* $\boldsymbol{P} = [\boldsymbol{p}_1, \ldots, \boldsymbol{p}_k]$*. Then,*

$$\langle \boldsymbol{\mathcal{W}}, \boldsymbol{x}^{\otimes m} \rangle = \hat{y}_{\mathcal{H}^m}(\boldsymbol{x}; \boldsymbol{\lambda}, \boldsymbol{P}) \tag{8}$$

$$\langle \boldsymbol{\mathcal{W}}, \boldsymbol{x}^{\otimes m} \rangle_{>} = \hat{y}_{\mathcal{A}^m}(\boldsymbol{x}; \boldsymbol{\lambda}, \boldsymbol{P}). \tag{9}$$

The result follows immediately from (5) and (6), and from the linearity of $\langle \cdot, \cdot \rangle$ and $\langle \cdot, \cdot \rangle_{>}$. Given $\boldsymbol{\mathcal{W}} \in \mathbb{S}^{d^m}$, let us define the following objective functions

$$L_{\mathcal{H}^m}(\boldsymbol{\mathcal{W}}) \coloneqq \sum_{i=1}^n \ell\left(y_i, \langle \boldsymbol{\mathcal{W}}, \boldsymbol{x}_i^{\otimes m} \rangle\right)$$

$$L_{\mathcal{A}^m}(\boldsymbol{\mathcal{W}}) \coloneqq \sum_{i=1}^n \ell\left(y_i, \langle \boldsymbol{\mathcal{W}}, \boldsymbol{x}_i^{\otimes m} \rangle_{>}\right).$$

If $\boldsymbol{\mathcal{W}}$ is decomposed as in (7), then from Lemma 5, we obtain $L_{\mathcal{K}}(\boldsymbol{\mathcal{W}}) = D_{\mathcal{K}}(\boldsymbol{\lambda}, \boldsymbol{P})$ for $\mathcal{K} = \mathcal{H}^m$ or $\mathcal{A}^m$. This

suggests that we can convert the problem of learning $\boldsymbol{\lambda}$ and $\boldsymbol{P}$ to that of learning a symmetric tensor $\boldsymbol{\mathcal{W}}$ of (symmetric) rank k. Thus, the problem of finding a small number of bases $\boldsymbol{p}_1, \dots, \boldsymbol{p}_k$ and their associated weights $\lambda_1, \dots, \lambda_k$ is converted to that of learning a *low-rank* symmetric tensor. Following (Candès et al., 2013), we call this approach *lifted*. Intuitively, we can think of $\boldsymbol{\mathcal{W}}$ as a tensor that contains the weights for predicting y of monomials of degree m. For instance, when $m = 3$, $\boldsymbol{\mathcal{W}}_{i,j,k}$ is the weight corresponding to the monomial $x_i x_j x_k$.

5.2. Multi-convex formulation

Estimating a low-rank symmetric tensor $\boldsymbol{\mathcal{W}} \in \mathbb{S}^{d^m}$ for arbitrary integer $m \geq 2$ is in itself a difficult non-convex problem. Nevertheless, based on a symmetrization trick, we can convert the problem to a multi-convex one, which we can easily minimize by alternating minimization. We first present our approach for the case $m = 2$ to give intuitions then explain how to extend it to $m \geq 3$.

Intuition with the second-order case. For the case $m = 2$, we need to estimate a low-rank symmetric matrix $\boldsymbol{W} \in \mathbb{S}^{d^2}$. Naively parameterizing $\boldsymbol{W} = \boldsymbol{P}\,\mathrm{diag}(\boldsymbol{\lambda})\boldsymbol{P}^{\mathrm{T}}$ and solving for $\boldsymbol{\lambda}$ and $\boldsymbol{P}$ does not lead to a multi-convex formulation for the case $\mathcal{K} = \mathcal{H}^2$. This is due to the fact that $\langle \boldsymbol{P}\,\mathrm{diag}(\boldsymbol{\lambda})\boldsymbol{P}^{\mathrm{T}}, \boldsymbol{x}^{\otimes 2}\rangle$ is *quadratic* in $\boldsymbol{P}$. Our key idea is to parametrize $\boldsymbol{W} = \mathcal{S}(\boldsymbol{U}\boldsymbol{V}^{\mathrm{T}})$ where $\boldsymbol{U}, \boldsymbol{V} \in \mathbb{R}^{d\times r}$ and $\mathcal{S}(\boldsymbol{M}) \coloneqq \frac{1}{2}(\boldsymbol{M} + \boldsymbol{M}^{\mathrm{T}}) \in \mathbb{S}^{d^2}$ is the *symmetrization* of $\boldsymbol{M} \in \mathbb{R}^{d^2}$. We then minimize $L_{\mathcal{K}}(\mathcal{S}(\boldsymbol{U}\boldsymbol{V}^{\mathrm{T}}))$ w.r.t. $\boldsymbol{U}, \boldsymbol{V}$.

The main advantage is that both $\langle \mathcal{S}(\boldsymbol{U}\boldsymbol{V}^{\mathrm{T}}), \cdot\rangle$ and $\langle \mathcal{S}(\boldsymbol{U}\boldsymbol{V}^{\mathrm{T}}), \cdot\rangle_>$ are *bi-linear* in $\boldsymbol{U}$ and $\boldsymbol{V}$. This implies that $L_{\mathcal{K}}(\mathcal{S}(\boldsymbol{U}\boldsymbol{V}^{\mathrm{T}}))$ is *bi-convex* in $\boldsymbol{U}$ and $\boldsymbol{V}$ and can therefore be efficiently minimized by alternating minimization. Once we obtained $\boldsymbol{W} = \mathcal{S}(\boldsymbol{U}\boldsymbol{V}^{\mathrm{T}})$, we can *optionally* compute its eigendecomposition $\boldsymbol{W} = \boldsymbol{P}\,\mathrm{diag}(\boldsymbol{\lambda})\boldsymbol{P}^{\mathrm{T}}$, with $k = \mathrm{rank}(\boldsymbol{W})$ and $r \leq k \leq 2r$, then apply (8) or (9) to obtain the model in kernel expansion form.

Extension to higher-order case. For $m \geq 3$, we now estimate a low-rank symmetric tensor $\boldsymbol{\mathcal{W}} = \mathcal{S}(\boldsymbol{\mathcal{M}}) \in \mathbb{S}^{d^m}$, where $\boldsymbol{\mathcal{M}} \in \mathbb{R}^{d^m}$ and $\mathcal{S}(\boldsymbol{\mathcal{M}})$ is the symmetrization of $\boldsymbol{\mathcal{M}}$ (cf. Appendix A.2). We decompose $\boldsymbol{\mathcal{M}}$ using m matrices of size $d \times r$. Let us call these matrices $\{\boldsymbol{U}^t\}_{t=1}^m$ and their columns $\boldsymbol{u}_s^t = [u_{1s}^t, \dots, u_{ds}^t]^{\mathrm{T}}$. Then the decomposition of $\boldsymbol{\mathcal{M}}$ can be expressed as a sum of rank-one tensors

$$\boldsymbol{\mathcal{M}} = \sum_{s=1}^{r} \boldsymbol{u}_s^1 \otimes \cdots \otimes \boldsymbol{u}_s^m. \tag{10}$$

Due to multi-linearity of (10) w.r.t. $\boldsymbol{U}^1, \dots, \boldsymbol{U}^m$, the objective function $L_{\mathcal{K}}$ is *multi-convex* in $\boldsymbol{U}^1, \dots, \boldsymbol{U}^m$.

Computing predictions efficiently. When $\mathcal{K} = \mathcal{H}^m$, predictions are computed by $\langle \boldsymbol{\mathcal{W}}, \boldsymbol{x}^{\otimes m}\rangle$. To compute them efficiently, we use the following lemma.

Lemma 6 *Symmetrization does not affect inner product*

$$\langle \mathcal{S}(\boldsymbol{\mathcal{M}}), \boldsymbol{\mathcal{X}}\rangle = \langle \boldsymbol{\mathcal{M}}, \boldsymbol{\mathcal{X}}\rangle \quad \forall\, \boldsymbol{\mathcal{M}} \in \mathbb{R}^{d^m}, \boldsymbol{\mathcal{X}} \in \mathbb{S}^{d^m}, m \geq 2. \tag{11}$$

Proof is given in Appendix A.2. Using $\boldsymbol{x}^{\otimes m} \in \mathbb{S}^{d^m}$, $\boldsymbol{\mathcal{W}} = \mathcal{S}(\boldsymbol{\mathcal{M}})$ and (10), we then obtain

$$\langle \boldsymbol{\mathcal{W}}, \boldsymbol{x}^{\otimes m}\rangle = \langle \boldsymbol{\mathcal{M}}, \boldsymbol{x}^{\otimes m}\rangle = \sum_{s=1}^{r}\prod_{t=1}^{m}\langle \boldsymbol{u}_s^t, \boldsymbol{x}\rangle.$$

As a result, we never need to explicitly compute the symmetrized tensor. For the case $\mathcal{K} = \mathcal{A}^2$, cf. Appendix D.3.

6. Regularization

In some applications, the number of bases or the rank constraint are not enough for obtaining good generalization performance and it is necessary to consider additional form of regularization. For the lifted objective with $\mathcal{K} = \mathcal{H}^2$ or $\mathcal{A}^2$, we use the typical Frobenius-norm regularization

$$\tilde{L}_{\mathcal{K}}(\boldsymbol{U}, \boldsymbol{V}) \coloneqq L_{\mathcal{K}}(\mathcal{S}(\boldsymbol{U}\boldsymbol{V}^{\mathrm{T}})) + \frac{\beta}{2}(\|\boldsymbol{U}\|_F^2 + \|\boldsymbol{V}\|_F^2), \tag{12}$$

where $\beta > 0$ is a regularization hyper-parameter. For the direct objective, we introduce the new regularization

$$\tilde{D}_{\mathcal{K}}(\boldsymbol{\lambda}, \boldsymbol{P}) \coloneqq D_{\mathcal{K}}(\boldsymbol{\lambda}, \boldsymbol{P}) + \beta \sum_{s=1}^{k} |\lambda_s|\, \|\boldsymbol{p}_s\|^2. \tag{13}$$

This allows us to regularize $\boldsymbol{\lambda}$ and $\boldsymbol{P}$ with a single hyper-parameter. Let us define the following nuclear norm penalized objective:

$$\bar{L}_{\mathcal{K}}(\boldsymbol{M}) \coloneqq L_{\mathcal{K}}(\mathcal{S}(\boldsymbol{M})) + \beta\|\boldsymbol{M}\|_*. \tag{14}$$

We can show that (12), (13) and (14) are equivalent in the following sense.

Theorem 2 *Equivalence of regularized problems*

Let $\mathcal{K} = \mathcal{H}^2$ or $\mathcal{A}^2$, then

$$\min_{\boldsymbol{M}\in\mathbb{R}^{d^2}} \bar{L}_{\mathcal{K}}(\boldsymbol{M}) = \min_{\substack{\boldsymbol{U}\in\mathbb{R}^{d\times r}\\ \boldsymbol{V}\in\mathbb{R}^{d\times r}}} \tilde{L}_{\mathcal{K}}(\boldsymbol{U}, \boldsymbol{V}) = \min_{\substack{\boldsymbol{\lambda}\in\mathbb{R}^{k}\\ \boldsymbol{P}\in\mathbb{R}^{d\times k}}} \tilde{D}_{\mathcal{K}}(\boldsymbol{\lambda}, \boldsymbol{P})$$

where $\mathrm{rank}(\boldsymbol{M}^*) \leq r = k$ *and* $\boldsymbol{M}^* \in \operatorname*{argmin}_{\boldsymbol{M}\in\mathbb{R}^{d^2}} \bar{L}_{\mathcal{K}}(\boldsymbol{M})$.

Proof is given in Appendix C. Our proof relies on the variational form of the nuclear norm and is thus limited to $m = 2$. One of the key ingredients of the proof is to show that the minimizer of (14) is always a symmetric matrix. In addition to Theorem 2, from (Abernethy et al., 2009), we also know that every local minimum $\boldsymbol{U}, \boldsymbol{V}$ of (12) gives a global solution $\boldsymbol{U}\boldsymbol{V}^{\mathrm{T}}$ of (14) provided that $\mathrm{rank}(\boldsymbol{M}^*) \leq r$. Proving a similar result for (13) is a future work. When $m \geq 3$, as used in our experiments, a squared Frobenius norm penalty on $\boldsymbol{P}$ (direct objective) or on $\{\boldsymbol{U}^t\}_{t=1}^m$ (lifted objective) works well in practice, although we lose the theoretical connection with the nuclear norm.

7. Coordinate descent algorithms

We now describe how to learn the model parameters by coordinate descent, which is a state-of-the-art *learning-rate free* solver for multi-convex problems (e.g., Yu et al. (2012)). In the following, we assume that ℓ is μ-smooth.

Direct objective with $\mathcal{K} = \mathcal{A}^m$ for $m \in \{2, 3\}$. First, we note that minimizing (13) w.r.t. $\boldsymbol{\lambda}$ can be reduced to a standard ℓ_1-regularized convex objective via a simple change of variable. Hence we focus on minimization w.r.t. $\boldsymbol{P}$.

Let us denote the elements of $\boldsymbol{P}$ by p_{js}. Then, our algorithm cyclically performs the following update for all $s \in [k]$ and $j \in [d]$:

$$p_{js} \leftarrow p_{js} - \eta^{-1} \left[\sum_{i=1}^{n} \ell'(y_i, \hat{y}_i) \frac{\partial \hat{y}_i}{\partial p_{js}} + 2\beta |\lambda_s| p_{js} \right],$$

where $\eta \coloneqq \mu \sum_{i=1}^{n} \left(\frac{\partial \hat{y}_i}{\partial p_{js}} \right)^2 + 2\beta|\lambda_s|$. Note that when ℓ is the squared loss, the above is equivalent to a Newton update and is the exact coordinate-wise minimizer.

The key challenge to use CD is computing $\frac{\partial \hat{y}_i}{\partial p_{js}} = \lambda_s \frac{\partial \mathcal{A}^m(\boldsymbol{p}_s, \boldsymbol{x}_i)}{\partial p_{js}}$ efficiently. Let us denote the elements of $\boldsymbol{X}$ by x_{ji}. Using Lemma 3, we obtain $\frac{\partial \mathcal{A}^2(\boldsymbol{p}_s, \boldsymbol{x}_i)}{\partial p_{js}} = \langle \boldsymbol{p}_s, \boldsymbol{x}_i \rangle x_{ji} - p_{js} x_{ji}^2$ and $\frac{\partial \mathcal{A}^3(\boldsymbol{p}_s, \boldsymbol{x}_i)}{\partial p_{js}} = \mathcal{A}^2(\boldsymbol{p}_s, \boldsymbol{x}_i) x_{ji} - p_{js} x_{ji}^2 \langle \boldsymbol{p}_s, \boldsymbol{x}_i \rangle + p_{js}^2 x_{ji}^3$. If for all $i \in [n]$ and for s fixed, we maintain $\langle \boldsymbol{p}_s, \boldsymbol{x}_i \rangle$ and $\mathcal{A}^2(\boldsymbol{p}_s, \boldsymbol{x}_i)$ (i.e., keep in sync after every update of p_{js}), then computing $\frac{\partial \hat{y}_i}{\partial p_{js}}$ takes $O(m)$ time. Hence the cost of one epoch, i.e. updating all elements of $\boldsymbol{P}$ once, is $O(mkn_z(\boldsymbol{X}))$. Complete details and pseudo code are given in Appendix D.1.

To our knowledge, this is the first CD algorithm capable of training third-order FMs. Supporting arbitrary $m \in \mathbb{N}$ is an important future work.

Lifted objective with $\mathcal{K} = \mathcal{H}^m$. Recall that we want to learn the matrices $\{\boldsymbol{U}^t\}_{t=1}^m$, whose columns we denote by $\boldsymbol{u}_s^t = [u_{1s}^t, \ldots, u_{ds}^t]^{\mathrm{T}}$. Our algorithm cyclically performs the following update for all $t \in [m]$, $s \in [r]$ and $j \in [d]$:

$$u_{js}^t \leftarrow u_{js}^t - \eta^{-1} \left[\sum_{i=1}^{n} \ell'(y_i, \hat{y}_i) \frac{\partial \hat{y}_i}{\partial u_{js}^t} + \beta u_{js}^t \right],$$

where $\eta \coloneqq \mu \sum_{i=1}^{n} \left(\frac{\partial \hat{y}_i}{\partial u_{js}^t} \right)^2 + \beta$. The main difficulty is computing $\frac{\partial \hat{y}_i}{\partial u_{js}^t} = \prod_{t' \neq t} \langle \boldsymbol{u}_s^{t'}, \boldsymbol{x}_i \rangle x_{ji}$ efficiently. If for all $i \in [n]$ and for t and s fixed, we maintain $\xi_i \coloneqq \prod_{t' \neq t} \langle \boldsymbol{u}_s^{t'}, \boldsymbol{x}_i \rangle$, then the cost of computing $\frac{\partial \hat{y}_i}{\partial u_{js}^t}$ is $O(1)$. Hence the cost of one epoch is $O(mrn_z(\boldsymbol{X}))$, the same as SGD. Complete details are given in Appendix D.2.

Convergence. The above updates decrease the objective *monotonically*. Convergence to a stationary point is guaranteed following (Bertsekas, 1999, Proposition 2.7.1).

8. Inhomogeneous polynomial models

The algorithms presented so far are designed for homogeneous polynomial kernels $\mathcal{H}^m$ and $\mathcal{A}^m$. These kernels only use monomials of the *same* degree m. However, in many applications, we would like to use monomials of *up to* some degree. In this section, we propose a simple idea to do so using the algorithms presented so far, unmodified. Our key observation is that we can easily turn homogeneous polynomials into inhomogeneous ones by augmenting the dimensions of the training data with dummy features.

We begin by explaining how to learn inhomogeneous polynomial models using $\mathcal{H}^m$. Let us denote $\tilde{\boldsymbol{p}}^{\mathrm{T}} \coloneqq [\gamma, \boldsymbol{p}^{\mathrm{T}}] \in \mathbb{R}^{d+1}$ and $\tilde{\boldsymbol{x}}^{\mathrm{T}} \coloneqq [1, \boldsymbol{x}^{\mathrm{T}}] \in \mathbb{R}^{d+1}$. Then, we obtain

$$\mathcal{H}^m(\tilde{\boldsymbol{p}}, \tilde{\boldsymbol{x}}) = \langle \tilde{\boldsymbol{p}}, \tilde{\boldsymbol{x}} \rangle^m = (\gamma + \langle \boldsymbol{p}, \boldsymbol{x} \rangle)^m = \mathcal{P}_\gamma^m(\boldsymbol{p}, \boldsymbol{x}).$$

Therefore, if we prepare the augmented training set $\tilde{\boldsymbol{x}}_1, \ldots, \tilde{\boldsymbol{x}}_n$, the problem of learning a model of the form $\sum_{s=1}^{k} \lambda_s \mathcal{P}_{\gamma_s}^m(\boldsymbol{p}_s, \boldsymbol{x})$ can be converted to that of learning a rank-k symmetric tensor $\boldsymbol{\mathcal{W}} \in \mathbb{S}^{(d+1)^m}$ using the method presented in Section 5. Note that the parameter γ_s is automatically learned from data for each basis $\boldsymbol{p}_s$.

Next, we explain how to learn inhomogeneous polynomial models using $\mathcal{A}^m$. Using Lemma 2, we immediately obtain for $1 \leq m \leq d$:

$$\mathcal{A}^m(\tilde{\boldsymbol{p}}, \tilde{\boldsymbol{x}}) = \mathcal{A}^m(\boldsymbol{p}, \boldsymbol{x}) + \gamma \mathcal{A}^{m-1}(\boldsymbol{p}, \boldsymbol{x}). \tag{15}$$

For instance, when $m = 2$, we obtain

$$\mathcal{A}^2(\tilde{\boldsymbol{p}}, \tilde{\boldsymbol{x}}) = \mathcal{A}^2(\boldsymbol{p}, \boldsymbol{x}) + \gamma \mathcal{A}^1(\boldsymbol{p}, \boldsymbol{x}) = \mathcal{A}^2(\boldsymbol{p}, \boldsymbol{x}) + \gamma \langle \boldsymbol{p}, \boldsymbol{x} \rangle.$$

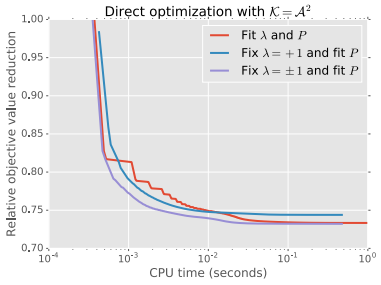
Therefore, if we prepare the augmented training set $\tilde{\boldsymbol{x}}_1, \ldots, \tilde{\boldsymbol{x}}_n$, we can easily learn a combination of linear kernel and second-order ANOVA kernel using methods presented in Section 4 or Section 5. Note that (15) only states the relation between two ANOVA kernels of consecutive degrees. Fortunately, we can also apply (15) recursively. Namely, by adding $m - 1$ dummy features, we can sum the kernels from $\mathcal{A}^m$ down to $\mathcal{A}^1$ (i.e., linear kernel).

9. Experimental results

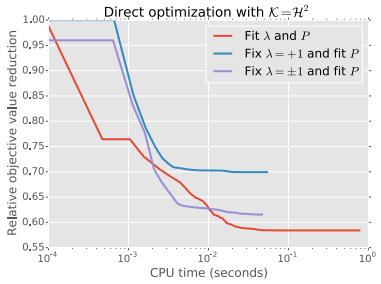
In this section, we present experimental results, focusing on regression tasks. Datasets are described in Appendix E. In all experiments, we set $\ell(y, \hat{y})$ to the squared loss.

9.1. Direct optimization: is it useful to fit $\boldsymbol{\lambda}$?

As explained in Section 4, there is no benefit to fitting $\boldsymbol{\lambda}$ when m is odd, since $\mathcal{A}^m$ and $\mathcal{H}^m$ can absorb $\boldsymbol{\lambda}$ into $\boldsymbol{P}$. This is however not the case when m is even: $\mathcal{A}^m$ and $\mathcal{H}^m$



(a) $\mathcal{K} = \mathcal{A}^2$



(b) $\mathcal{K} = \mathcal{H}^2$

Figure 1. **Effect of fitting $\boldsymbol{\lambda}$ when using direct optimization** on the *diabetes* dataset with $m = 2$, $\beta = 10$ and $k = 4$. Objective values were computed by (13) and were normalized by the worst initialization's objective value.

can absorb absolute values but not negative signs (unless complex numbers are allowed for parameters). Therefore, when m is even, the class of functions we can represent with models of the form (1) is possibly smaller if we fix $\boldsymbol{\lambda} = \mathbf{1}$ (as done in FMs).

To check that this is indeed the case, on the *diabetes* dataset, we minimized (13) with $m = 2$ as follows:

a) minimize w.r.t. both $\boldsymbol{\lambda}$ and $\boldsymbol{P}$ alternatingly,
b) fix $\lambda_s = 1$ for $s \in [k]$ and minimize w.r.t. $\boldsymbol{P}$,
c) fix $\lambda_s = \pm 1$ with proba. 0.5 and minimize w.r.t. $\boldsymbol{P}$.

We initialized elements of $\boldsymbol{P}$ by $p_{js} \sim \mathcal{N}(0, 0.01)$ for all $j \in [d]$, $s \in [k]$. Our results are shown in Figure 1. For $\mathcal{K} = \mathcal{A}^2$, we use CD and for $\mathcal{K} = \mathcal{H}^2$, we use L-BFGS. Note that since (13) is convex w.r.t. $\boldsymbol{\lambda}$, a) is insensitive to the initialization of $\boldsymbol{\lambda}$ as long as we fit $\boldsymbol{\lambda}$ before $\boldsymbol{P}$. Not surprisingly, fitting $\boldsymbol{\lambda}$ allows us to achieve a smaller objective value. This is especially apparent when $\mathcal{K} = \mathcal{H}^2$. However, the difference is much smaller when $\mathcal{K} = \mathcal{A}^2$. We give intuitions as to why this is the case in Section 10.

We emphasize that this experiment was designed to confirm that fitting $\boldsymbol{\lambda}$ does indeed improve representation power of the model when m is even. In practice, it is possible that fixing $\boldsymbol{\lambda} = \mathbf{1}$ reduces overfitting and thus improves *generalization error*. However, this highly depends on the data.

9.2. Direct vs. lifted optimization

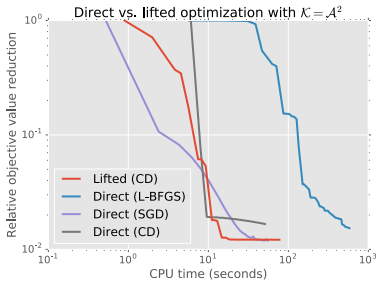
In this section, we compare the direct and lifted optimization approaches on high-dimensional data when $m = 2$. To compare the two approaches fairly, we propose the following initialization scheme. Recall that, at the end of the day, both approaches are essentially learning a low rank symmetric matrix: $\boldsymbol{W} = \mathcal{S}(\boldsymbol{U}\boldsymbol{V}^{\mathrm{T}})$ for lifted and $\boldsymbol{W} = \boldsymbol{P}\,\mathrm{diag}(\boldsymbol{\lambda})\boldsymbol{P}^{\mathrm{T}}$ for direct optimization. This suggests that we can easily convert the matrices $\boldsymbol{U}, \boldsymbol{V} \in \mathbb{R}^{d\times r}$ used for initializing lifted optimization to $\boldsymbol{P} \in \mathbb{R}^{d\times k}$ and $\boldsymbol{\lambda} \in \mathbb{R}^{d\times k}$ by computing the (reduced) eigendecomposition of $\mathcal{S}(\boldsymbol{U}\boldsymbol{V}^{\mathrm{T}})$. Note that because we solve the lifted optimization problem by coordinate descent, $\boldsymbol{U}\boldsymbol{V}^{\mathrm{T}}$ is never symmetric and therefore the rank of $\mathcal{S}(\boldsymbol{U}\boldsymbol{V}^{\mathrm{T}})$ is usually twice that of $\boldsymbol{U}\boldsymbol{V}^{\mathrm{T}}$. Hence, in practice, we have that $r = k/2$. In our experiment, we compared four methods: lifted objective solved by CD, direct objective solved by CD, L-BFGS and SGD. For lifted optimization, we initialized the elements of $\boldsymbol{U}$ and $\boldsymbol{V}$ by sampling from $\mathcal{N}(0, 0.01)$. For direct optimization, we obtained $\boldsymbol{P}$ and $\boldsymbol{\lambda}$ as explained. Results on the *E2006-tfidf* high-dimensional dataset are shown in Figure 2. For $\mathcal{K} = \mathcal{A}^2$, we find that Lifted (CD) and Direct (CD) have similar convergence speed and both outperform Direct (L-BFGS). For $\mathcal{K} = \mathcal{H}^2$, we find that Lifted (CD) outperforms both Direct (L-BFGS) and Direct (SGD). Note that we did not implement Direct (CD) for $\mathcal{K} = \mathcal{H}^2$ since the direct optimization problem is not coordinate-wise convex, as explained in Section 5.

9.3. Recommender system experiment

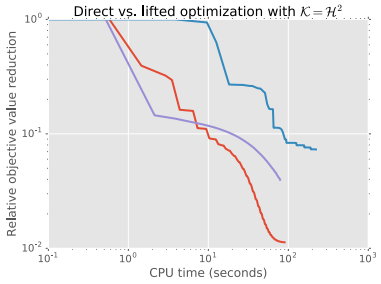
To confirm the ability of the proposed framework to infer the weights of unobserved feature interactions, we conducted experiments on *Last.fm* and *Movielens 1M*, two standard recommender system datasets. Following (Rendle, 2012), matrix factorization can be reduced to FMs by creating a dataset of $(\boldsymbol{x}_i, y_i)$ pairs where $\boldsymbol{x}_i$ contains the one-hot encoding of the user and item and y_i is the corresponding rating (i.e., number of training instances equals number of ratings). We compared four models:

a) $\mathcal{K} = \mathcal{A}^2$ (augment): $\hat{y} = \hat{y}_{\mathcal{A}^2}(\tilde{\boldsymbol{x}})$, with $\tilde{\boldsymbol{x}}^{\mathrm{T}} := [1, \boldsymbol{x}^{\mathrm{T}}]$,
b) $\mathcal{K} = \mathcal{A}^2$ (linear combination): $\hat{y} = \langle \boldsymbol{w}, \boldsymbol{x} \rangle + \hat{y}_{\mathcal{A}^2}(\boldsymbol{x})$,
c) $\mathcal{K} = \mathcal{H}^2$ (augment): $\hat{y} = \hat{y}_{\mathcal{H}^2}(\tilde{\boldsymbol{x}})$ and
d) $\mathcal{K} = \mathcal{H}^2$ (linear combination): $\hat{y} = \langle \boldsymbol{w}, \boldsymbol{x} \rangle + \hat{y}_{\mathcal{H}^2}(\boldsymbol{x})$,

where $\boldsymbol{w} \in \mathbb{R}^d$ is a vector of first-order weights, estimated from training data. Note that b) and d) are exactly the same as FMs and PNs, respectively. Results are shown in Figure 3. We see that $\mathcal{A}^2$ tends to outperform $\mathcal{H}^2$ on these tasks. We hypothesize that this the case because features are binary (cf., discussion in Section 10). We also see that simply augmenting the features as suggested in Section 8 is comparable or better than learning additional first-order feature weights, as done in FMs and PNs.

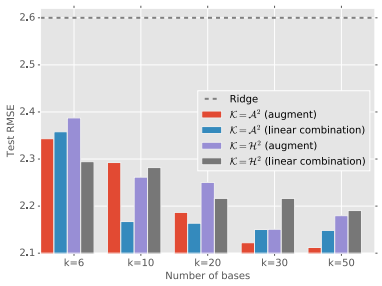


(a) $\mathcal{K} = \mathcal{A}^2$

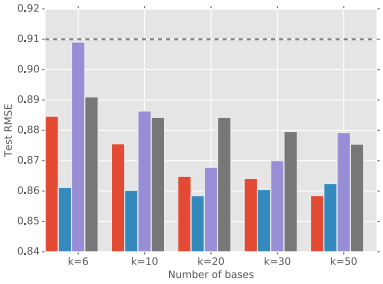


(b) $\mathcal{K} = \mathcal{H}^2$

Figure 2. **Comparison of the direct and lifted optimization approaches** on the *E2006-tfidf* high-dimensional dataset with $m = 2$, $\beta = 100$ and $r = k/2 = 10$. In order to learn an inhomogeneous polynomial, we added a dummy feature to all training instances, as explained in Section 8. Objective values shown were computed by (13) and (12) and normalized by the initialization's objective value.



(a) Last.fm



(b) Movielens 1M

Figure 3. **Predicted rating error** on the *Last.fm* and *Movielens 1M* datasets. The metric used is RMSE on the test set (lower is better). The hyper-parameter β was selected from 10 log-spaced values in the interval $[10^{-3}, 10^{3}]$ by 5-fold cross-validation.

9.4. Low-budget non-linear regression experiment

In this experiment, we demonstrate the ability of the proposed framework to reach good regression performance with a small number of bases k. We compared:

a) Proposed with $\mathcal{K} = \mathcal{H}^3$ (with augmented features),
b) Proposed with $\mathcal{K} = \mathcal{A}^3$ (with augmented features),
c) Nyström method with $\mathcal{K} = \mathcal{P}_1^3$ and
d) Random Selection: choose $\boldsymbol{p}_1, \ldots, \boldsymbol{p}_k$ uniformly at random from training set and use $\mathcal{K} = \mathcal{P}_1^3$.

For a) and b) we used the lifted approach. For fair comparison in terms of model size (number of floats used), we set $r = k/3$. Results on the *abalone*, *cadata* and *cpusmall* datasets are shown in Figure 4. We see that *i)* the proposed framework reaches the same performance as kernel ridge regression with much fewer bases than other methods and *ii)* $\mathcal{H}^3$ tends to outperform $\mathcal{A}^3$ on these tasks. Similar trends were observed when using $\mathcal{K} = \mathcal{H}^2$ or $\mathcal{A}^2$.

10. Discussion

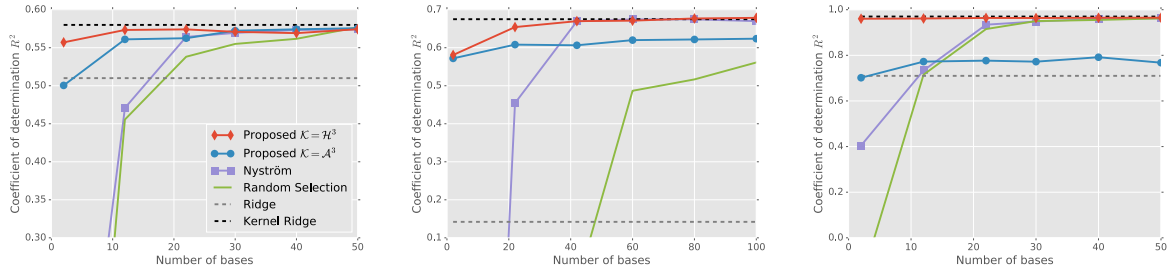
Ability to infer weights of unobserved interactions. In our view, one of the strengths of PNs and FMs is their ability to infer the weights of unobserved feature interactions, unlike traditional kernel methods. To see why, recall that in kernel methods, predictions are computed by $\hat{y} = \sum_{i=1}^{n} \alpha_i \mathcal{K}(\boldsymbol{x}_i, \boldsymbol{x})$. When $\mathcal{K} = \mathcal{H}^m$ or $\mathcal{A}^m$, by Lemma 5, this is equivalent to $\hat{y} = \langle \widetilde{\boldsymbol{\mathcal{W}}}, \boldsymbol{x}^{\otimes m} \rangle$ or $\langle \widetilde{\boldsymbol{\mathcal{W}}}, \boldsymbol{x}^{\otimes m} \rangle_{>}$ if we set $\widetilde{\boldsymbol{\mathcal{W}}} \coloneqq \sum_{i=1}^{n} \alpha_i \boldsymbol{x}_i^{\otimes m}$. Thus, in kernel methods, the weight associated with $x_{j_1} \ldots x_{j_m}$ can be written as a linear combination of the training data's monomials:

$$\widetilde{\boldsymbol{\mathcal{W}}}_{j_1,\ldots,j_m} = \sum_{i=1}^{n} \alpha_i x_{j_1 i} \ldots x_{j_m i}.$$

Assuming binary features, the weights of monomials that were never observed in the training set are zero. In contrast, in PNs and FMs, we have $\boldsymbol{\mathcal{W}} = \sum_{s=1}^{k} \lambda_i \boldsymbol{p}_s^{\otimes m}$ and therefore the weight associated with $x_{j_1} \ldots x_{j_m}$ becomes

$$\boldsymbol{\mathcal{W}}_{j_1,\ldots,j_m} = \sum_{s=1}^{k} \lambda_s p_{j_1 s} \ldots p_{j_m s}.$$

Because parameters are shared across monomials, PNs and FMs are able to *interpolate* the weights of monomials that were never observed in the training set. This is the key property which makes it possible to use them on recommender system tasks. In future work, we plan to apply PNs and FMs to biological data, where this property should be



(a) abalone (b) cadata (c) cpusmall

Figure 4. **Regression performance as a function of the number of bases.** The metric used is the coefficient of determination on the test set (higher is better). Regularization parameter was selected from 10 log-spaced values in $[10^{-4}, 10^{4}]$ by 5-fold cross-validation.

very useful, e.g., for inferring higher-order interactions between genes.

ANOVA kernel vs. polynomial kernel. One of the key properties of the ANOVA kernel $\mathcal{A}^m(\boldsymbol{p}, \boldsymbol{x})$ is multi-linearity w.r.t. elements of $\boldsymbol{p}$ (Lemma 2). This is the key difference with $\mathcal{H}^m(\boldsymbol{p}, \boldsymbol{x})$ which makes the direct optimization objective multi-convex when $\mathcal{K} = \mathcal{A}^m$ (Theorem 1). However, because we need to ignore irrelevant monomials, computing the kernel and its gradient is more challenging. Deriving efficient training algorithms for arbitrary $m \in \mathbb{N}$ is an important future work.

In our experiments in Section 9.1, we showed that fixing $\boldsymbol{\lambda} = \mathbf{1}$ works relatively well when $\mathcal{K} = \mathcal{A}^2$. To see intuitively why this is the case, note that fixing $\boldsymbol{\lambda} = \mathbf{1}$ is equivalent to constraining the weight matrix $\boldsymbol{W}$ to be positive semidefinite, i.e., $\exists \boldsymbol{P}$ s.t. $\boldsymbol{W} = \boldsymbol{P}\boldsymbol{P}^{\mathrm{T}}$. Next, observe that we can rewrite the prediction function as

$$\hat{y}_{\mathcal{A}^2}(\boldsymbol{x}; \mathbf{1}, \boldsymbol{P}) = \langle \boldsymbol{P}\boldsymbol{P}^{\mathrm{T}}, \boldsymbol{x}^{\otimes 2} \rangle_{>} = \langle \mathcal{U}(\boldsymbol{P}\boldsymbol{P}^{\mathrm{T}}), \boldsymbol{x}^{\otimes 2} \rangle,$$

where $\mathcal{U}(\boldsymbol{M})$ is a mask which sets diagonal and lower-diagonal elements of $\boldsymbol{M}$ to zero. We therefore see that when using $\mathcal{K} = \mathcal{A}^2$, we are learning a *strictly upper-triangular matrix*, parametrized by $\boldsymbol{P}\boldsymbol{P}^{\mathrm{T}}$. Importantly, the matrix $\mathcal{U}(\boldsymbol{P}\boldsymbol{P}^{\mathrm{T}})$ is not positive semidefinite. This is what gives the model some degree of freedom, even though $\boldsymbol{P}\boldsymbol{P}^{\mathrm{T}}$ is positive semidefinite. In contrast, when using $\mathcal{K} = \mathcal{H}^2$, if we fix $\boldsymbol{\lambda} = \mathbf{1}$, then we have that

$$\hat{y}_{\mathcal{H}^2}(\boldsymbol{x}; \mathbf{1}, \boldsymbol{P}) = \langle \boldsymbol{P}\boldsymbol{P}^{\mathrm{T}}, \boldsymbol{x}^{\otimes 2} \rangle = \boldsymbol{x}^{\mathrm{T}} \boldsymbol{P}\boldsymbol{P}^{\mathrm{T}} \boldsymbol{x} \geq 0$$

and therefore the model is unable to predict negative values.

Empirically, we showed in Section 9.4 that $\mathcal{H}^m$ outperforms $\mathcal{A}^m$ for low-budget non-linear regression. In contrast, we showed in Section 9.3 that $\mathcal{A}^m$ outperforms $\mathcal{H}^m$ for recommender systems. The main difference between the two experiments is the nature of the features used: continuous for the former and binary for the latter. For binary features, squared features $x_1^2, \ldots, x_d^2$ are redundant with $x_1, \ldots, x_d$ and are therefore not expected to help improve accuracy. On the contrary, they might introduce bias towards first-order features. We hypothesize that the ANOVA kernel is in general a better choice for binary features, although this needs to be verified by more experiments, for instance on natural language processing (NLP) tasks.

Direct vs. lifted optimization. The main advantage of direct optimization is that we only need to estimate $\boldsymbol{\lambda} \in \mathbb{R}^k$ and $\boldsymbol{P} \in \mathbb{R}^{d \times k}$ and therefore the number of parameters to estimate is independent of the degree m. Unfortunately, the approach is neither convex nor multi-convex when using $\mathcal{K} = \mathcal{H}^m$. In addition, the regularized objective (13) is non-smooth w.r.t. $\boldsymbol{\lambda}$. In Section 5, we proposed to reformulate the problem as one of low-rank symmetric tensor estimation and used a symmetrization trick to obtain a multi-convex smooth objective function. Because this objective involves the estimation of m matrices of size $d \times r$, we need to set $r = k/m$ for fair comparison with the direct objective in terms of model size. When $\mathcal{K} = \mathcal{A}^m$, we showed that the direct objective is readily multi-convex. However, an advantage of our lifted objective when $\mathcal{K} = \mathcal{A}^m$ is that it is convex w.r.t. larger block of variables than the direct objective.

11. Conclusion

In this paper, we revisited polynomial networks (Livni et al., 2014) and factorization machines (Rendle, 2010; 2012) from a unified perspective. We proposed direct and lifted optimization approaches and showed their equivalence in the regularized case for $m = 2$. With respect to PNs, we proposed the first CD solver with support for arbitrary integer $m \geq 2$. With respect to FMs, we made several novel contributions including making a connection with the ANOVA kernel, proving important properties of the objective function and deriving the first CD solver for third-order FMs. Empirically, we showed that the proposed algorithms achieve excellent performance on non-linear regression and recommender system tasks.

Acknowledgments

This work was partially conducted as part of "Research and Development on Fundamental and Applied Technologies for Social Big Data", commissioned by the National Institute of Information and Communications Technology (NICT), Japan. We also thank Vlad Niculae, Olivier Grisel, Fabian Pedregosa and Joseph Salmon for their valuable comments.

References

Abernethy, Jacob, Bach, Francis, Evgeniou, Theodoros, and Vert, Jean-Philippe. A new approach to collaborative filtering: Operator estimation with spectral regularization. *J. Mach. Learn. Res.*, 10:803–826, 2009.

Avron, Haim, Nguyen, Huy, and Woodruff, David. Subspace embeddings for the polynomial kernel. In *Advances in Neural Information Processing Systems 27*, pp. 2258–2266. 2014.

Bertsekas, Dimitri P. *Nonlinear programming*. Athena scientific Belmont, 1999.

Blondel, Mathieu, Fujino, Akinori, and Ueada, Naonori. Convex factorization machines. In *Proceedings of European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML PKDD)*, 2015.

Candès, Emmanuel J., Eldar, Yonina C., Strohmer, Thomas, and Voroninski, Vladislav. Phase retrieval via matrix completion. *SIAM Journal on Imaging Sciences*, 6(1):199–225, 2013.

Chang, Yin-Wen, Hsieh, Cho-Jui, Chang, Kai-Wei, Ringgaard, Michael, and Lin, Chih-Jen. Training and testing low-degree polynomial data mappings via linear svm. *Journal of Machine Learning Research*, 11:1471–1490, 2010.

Chen, Danqi and Manning, Christopher D. A fast and accurate dependency parser using neural networks. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, volume 1, pp. 740–750, 2014.

Comon, Pierre, Golub, Gene, Lim, Lek-Heng, and Mourrain, Bernard. Symmetric tensors and symmetric tensor rank. *SIAM Journal on Matrix Analysis and Applications*, 30(3):1254–1279, 2008.

Kar, Purushottam and Karnick, Harish. Random feature maps for dot product kernels. In *Proceedings of the International Conference on Artificial Intelligence and Statistics*, pp. 583–591, 2012.

Livni, Roi, Shalev-Shwartz, Shai, and Shamir, Ohad. On the computational efficiency of training neural networks. In *Advances in Neural Information Processing Systems*, pp. 855–863, 2014.

Mazumder, Rahul, Hastie, Trevor, and Tibshirani, Robert. Spectral regularization algorithms for learning large incomplete matrices. *The Journal of Machine Learning Research*, 11:2287–2322, 2010.

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

Pham, Ninh and Pagh, Rasmus. Fast and scalable polynomial kernels via explicit feature maps. In *Proceedings of the 19th KDD conference*, pp. 239–247, 2013.

Rendle, Steffen. Factorization machines. In *Proceedings of International Conference on Data Mining*, pp. 995–1000. IEEE, 2010.

Rendle, Steffen. Factorization machines with libfm. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 3(3):57–78, 2012.

Shawe-Taylor, John and Cristianini, Nello. *Kernel Methods for Pattern Analysis*. Cambridge University Press, 2004.

Sonnenburg, Sören and Franc, Vojtech. Coffin: A computational framework for linear svms. In *Proceedings of the 27th International Conference on Machine Learning*, pp. 999–1006, 2010.

Stitson, Mark, Gammerman, Alex, Vapnik, Vladimir, Vovk, Volodya, Watkins, Chris, and Weston, Jason. Support vector regression with anova decomposition kernels. Technical report, Royal Holloway University of London, 1997.

Vapnik, Vladimir. *Statistical learning theory*. Wiley, 1998.

Wang, Z., Crammer, K., and Vucetic, S. Multi-class pegasos on a budget. In *Proceedings of the 27th International Conference on Machine Learning (ICML)*, pp. 1143–1150, 2010.

Williams, Christopher K. I. and Seeger, Matthias. Using the nyström method to speed up kernel machines. In *Advances in Neural Information Processing Systems 13*, pp. 682–688. 2001.

Yu, Hsiang-Fu, Hsieh, Cho-Jui, Si, Si, and Dhillon, Inderjit S. Scalable coordinate descent approaches to parallel matrix factorization for recommender systems. In *ICDM*, pp. 765–774, 2012.

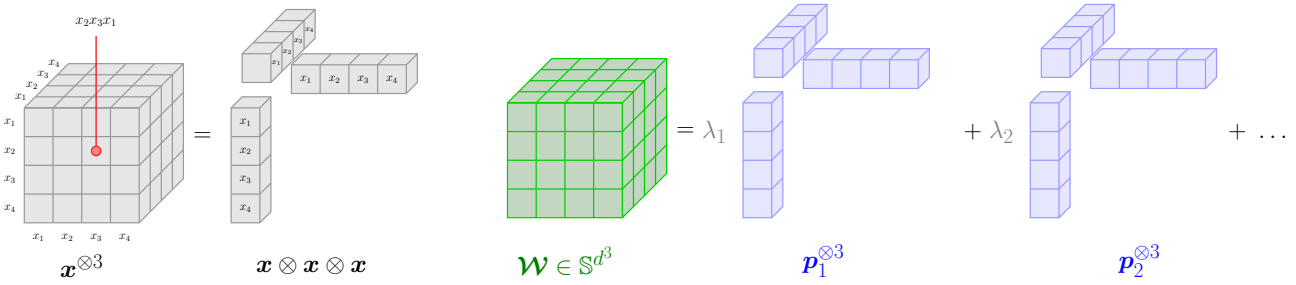


Figure 5. Illustration of symmetric rank-one tensor (left) and symmetric outer product decomposition (right).

Supplementary material

A. Symmetric tensors

A.1. Background

Let $\mathbb{R}^{d_1 \times \cdots \times d_m}$ be the set of $d_1 \times \cdots \times d_m$ real m-order tensors. In this paper, we focus on cubical tensors, i.e., $d_1 = \cdots = d_m = d$. We denote the set of m-order cubical tensors by $\mathbb{R}^{d^m}$. We denote the elements of $\boldsymbol{\mathcal{M}} \in \mathbb{R}^{d^m}$ by $\boldsymbol{\mathcal{M}}_{j_1,\ldots,j_m}$, where $j_1, \ldots, j_m \in [d]$.

Let $\boldsymbol{\sigma} = [\sigma_1, \ldots, \sigma_m]$ be a permutation of $\{1, \ldots, m\}$. Given $\boldsymbol{\mathcal{M}} \in \mathbb{R}^{d^m}$, we define $\boldsymbol{\mathcal{M}}_{\boldsymbol{\sigma}} \in \mathbb{R}^{d^m}$ as the tensor such that

$$(\boldsymbol{\mathcal{M}}_{\boldsymbol{\sigma}})_{j_1,\ldots,j_m} \coloneqq \boldsymbol{\mathcal{M}}_{j_{\sigma_1},\ldots,j_{\sigma_m}} \quad \forall j_1, \ldots, j_m \in [d].$$

In other words $\boldsymbol{\mathcal{M}}_{\boldsymbol{\sigma}}$ is a copy of $\boldsymbol{\mathcal{M}}$ with its axes permuted. This generalizes the concept of transpose to tensors.

Let P_m be the set of all permutations of $\{1, \ldots, m\}$. We say that a tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{d^m}$ is symmetric if and only if

$$\boldsymbol{\mathcal{X}}_{\boldsymbol{\sigma}} = \boldsymbol{\mathcal{X}} \quad \forall \boldsymbol{\sigma} \in P_m.$$

We denote the set of symmetric tensors by $\mathbb{S}^{d^m}$.

Given $\boldsymbol{\mathcal{M}} \in \mathbb{R}^{d^m}$, we define the symmetrization of $\boldsymbol{\mathcal{M}}$ by

$$\mathcal{S}(\boldsymbol{\mathcal{M}}) = \frac{1}{m!} \sum_{\boldsymbol{\sigma} \in P_m} \boldsymbol{\mathcal{M}}_{\boldsymbol{\sigma}}.$$

Note that when $m = 2$, then $\mathcal{S}(\boldsymbol{M}) = \frac{1}{2}(\boldsymbol{M} + \boldsymbol{M}^{\mathrm{T}})$.

Given $\boldsymbol{x} \in \mathbb{R}^d$, we define a symmetric rank-one tensor by $\boldsymbol{x}^{\otimes m} \coloneqq \underbrace{\boldsymbol{x} \otimes \cdots \otimes \boldsymbol{x}}_{m \text{ times}} \in \mathbb{S}^{d^m}$, i.e., $(\boldsymbol{x}^{\otimes m})_{j_1,j_2,\ldots,j_m} = x_{j_1} x_{j_2} \ldots x_{j_m}$. We denote the symmetric outer product decomposition (Comon et al., 2008) of $\boldsymbol{\mathcal{W}} \in \mathbb{S}^{d^m}$ by

$$\boldsymbol{\mathcal{W}} = \sum_{s=1}^{k} \lambda_s \boldsymbol{p}_s^{\otimes m},$$

where k is called the symmetric rank of $\boldsymbol{\mathcal{W}}$. This generalizes the concept of eigendecomposition to tensors. These two concepts are illustrated in Figure 5.

A.2. Proof of Lemma 6

Assume $\boldsymbol{\mathcal{M}} \in \mathbb{R}^{d^m}$ and $\boldsymbol{\mathcal{X}} \in \mathbb{S}^{d^m}$. Then,

$$\begin{aligned}
\langle \mathcal{S}(\boldsymbol{\mathcal{M}}), \boldsymbol{\mathcal{X}} \rangle &= \frac{1}{m!} \sum_{\boldsymbol{\sigma} \in P_m} \langle \boldsymbol{\mathcal{M}}_{\boldsymbol{\sigma}}, \boldsymbol{\mathcal{X}} \rangle && \text{by definition of } \mathcal{S}(\boldsymbol{\mathcal{M}}) \text{ and by linearity} \\
&= \frac{1}{m!} \sum_{\boldsymbol{\sigma} \in P_m} \langle (\boldsymbol{\mathcal{M}}_{\boldsymbol{\sigma}})_{\boldsymbol{\sigma}^{-1}}, \boldsymbol{\mathcal{X}}_{\boldsymbol{\sigma}^{-1}} \rangle && \text{since } \langle \boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{B}} \rangle = \langle \boldsymbol{\mathcal{A}}_{\boldsymbol{\sigma}}, \boldsymbol{\mathcal{B}}_{\boldsymbol{\sigma}} \rangle \; \forall \boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{B}} \in \mathbb{R}^{d^m}, \forall \boldsymbol{\sigma} \in P_m \\
&= \frac{1}{m!} \sum_{\boldsymbol{\sigma} \in P_m} \langle \boldsymbol{\mathcal{M}}, \boldsymbol{\mathcal{X}}_{\boldsymbol{\sigma}^{-1}} \rangle && \text{by definition of inverse permutation} \\
&= \frac{1}{m!} \sum_{\boldsymbol{\sigma} \in P_m} \langle \boldsymbol{\mathcal{M}}, \boldsymbol{\mathcal{X}} \rangle && \text{since } \boldsymbol{\mathcal{X}} \in \mathbb{S}^{d^m} \\
&= \langle \boldsymbol{\mathcal{M}}, \boldsymbol{\mathcal{X}} \rangle.
\end{aligned}$$

B. Proofs related to ANOVA kernels

B.1. Proof of multi-linearity (Lemma 2)

For $m = 1$, we have

$$\begin{aligned}
\mathcal{A}^1(\boldsymbol{p}, \boldsymbol{x}) &= \sum_{j=1}^{d} p_j x_j \\
&= \sum_{k \neq j} p_k x_k + p_j x_j \\
&= \mathcal{A}^1(\boldsymbol{p}_{\neg j}, \boldsymbol{x}_{\neg j}) + \; p_j x_j \; \mathcal{A}^0(\boldsymbol{p}_{\neg j}, \boldsymbol{x}_{\neg j})
\end{aligned}$$

where we used $\mathcal{A}^0(\boldsymbol{p}, \boldsymbol{x}) = 1$.

For $1 < m \leq d$, first notice that we can rewrite (3) as

$$\begin{aligned}
\mathcal{A}^m(\boldsymbol{p}, \boldsymbol{x}) &= \sum_{j_m > \cdots > j_1} p_{j_1} x_{j_1} \dots p_{j_m} x_{j_m} \quad j_k \in [d], k \in [m] \\
&= \sum_{j_1=1}^{d-m+1} \sum_{j_2=j_1+1}^{d-m+2} \cdots \sum_{j_m=j_{m-1}+1}^{d} p_{j_1} x_{j_1} \dots p_{j_m} x_{j_m}.
\end{aligned}$$

Then,

$$\begin{aligned}
\mathcal{A}^m(\boldsymbol{p}, \boldsymbol{x}) &= \sum_{j_1=1}^{d-m+1} \sum_{j_2=j_1+1}^{d-m+2} \cdots \sum_{j_m=j_{m-1}+1}^{d} p_{j_1} x_{j_1} p_{j_2} x_{j_2} \dots p_{j_m} x_{j_m} \\
&= \sum_{j_2=j_1+1}^{d-m+2} \cdots \sum_{j_m=j_{m-1}+1}^{d} p_1 x_1 p_{j_2} x_{j_2} \dots p_{j_m} x_{j_m} + \\
&\quad \sum_{j_1=2}^{d-m+1} \sum_{j_2=j_1+1}^{d-m+2} \cdots \sum_{j_m=j_{m-1}+1}^{d} p_{j_1} x_{j_1} p_{j_2} x_{j_2} \dots p_{j_m} x_{j_m} \\
&= p_1 x_1 \mathcal{A}^{m-1}(\boldsymbol{p}_{\neg 1}, \boldsymbol{x}_{\neg 1}) + \mathcal{A}^m(\boldsymbol{p}_{\neg 1}, \boldsymbol{x}_{\neg 1}).
\end{aligned}$$

We can always permute the elements of $\boldsymbol{p}$ and $\boldsymbol{x}$ without changing $\mathcal{A}^m(\boldsymbol{p}, \boldsymbol{x})$. It follows that

$$\mathcal{A}^m(\boldsymbol{p}, \boldsymbol{x}) = p_j x_j \mathcal{A}^{m-1}(\boldsymbol{p}_{\neg j}, \boldsymbol{x}_{\neg j}) + \mathcal{A}^m(\boldsymbol{p}_{\neg j}, \boldsymbol{x}_{\neg j}) \quad \forall j \in [d].$$

B.2. Efficient computation when $m \in \{2, 3\}$

Using the multinomial theorem, we can expand the homogeneous polynomial kernel as

$$\mathcal{H}^m(\boldsymbol{p}, \boldsymbol{x}) = \langle \boldsymbol{p}, \boldsymbol{x} \rangle^m = \sum_{k_1 + \cdots + k_d = m} \binom{m}{k_1, \ldots, k_d} \prod_{j=1}^{d} (p_j x_j)^{k_j} \tag{16}$$

where

$$\binom{m}{k_1, \ldots, k_d} \coloneqq \frac{m!}{k_1! \ldots k_d!}$$

is the multinomial coefficient and $k_j \in \{0, 1, \ldots, m\}$. Intuitively, $\binom{m}{k_1, \ldots, k_d}$ is the weight of the monomial $(p_1 x_1)^{k_1} \ldots (p_d x_d)^{k_d}$ in the expansion. For instance, if $\boldsymbol{p}, \boldsymbol{x} \in \mathbb{R}^3$, then the weight of $p_1 x_1 p_3^2 x_3^2$ is $\binom{3}{1,0,2} = 3$. The main observation is that monomials where all $k_1, \ldots, k_d$ are in $\{0, 1\}$ correspond to monomials of (3). If we can compute all other monomials efficiently, then we just need to subtract them from the homogeneous kernel in order to obtain (3).

To simplify notation, we define the shorthands

$$\rho_j \coloneqq p_j x_j, \quad \mathcal{D}^m(\boldsymbol{p}, \boldsymbol{x}) \coloneqq \sum_{j=1}^{d} \rho_j^m \quad \text{and} \quad \mathcal{D}^{m,n}(\boldsymbol{p}, \boldsymbol{x}) \coloneqq \mathcal{D}^m(\boldsymbol{p}, \boldsymbol{x}) \mathcal{D}^n(\boldsymbol{p}, \boldsymbol{x}).$$

Case $m = 2$

For $m = 2$, the possible monomials are of the form ρ_j^2 for all j and $\rho_i \rho_j$ for $j > i$. Applying (16), we obtain

$$\begin{aligned} \mathcal{H}^2(\boldsymbol{p}, \boldsymbol{x}) &= \sum_{j=1}^{d} \rho_j^2 + 2 \sum_{j>i} \rho_i \rho_j \\ &= \mathcal{D}^2(\boldsymbol{p}, \boldsymbol{x}) + 2\mathcal{A}^2(\boldsymbol{p}, \boldsymbol{x}) \end{aligned}$$

and therefore

$$\mathcal{A}^2(\boldsymbol{p}, \boldsymbol{x}) = \frac{1}{2} \left[\mathcal{H}^2(\boldsymbol{p}, \boldsymbol{x}) - \mathcal{D}^2(\boldsymbol{p}, \boldsymbol{x}) \right].$$

This formula was already mentioned in (Stitson et al., 1997). It was also rediscovered in (Rendle, 2010; 2012), although the connection with the ANOVA kernel was not identified.

Case $m = 3$

For $m = 3$, the possible monomials are of the form ρ_j^3 for all j, $\rho_i \rho_j^2$ for $i \neq j$ and $\rho_i \rho_j \rho_k$ for $k > j > i$. Applying (16), we obtain

$$\begin{aligned} \mathcal{H}^3(\boldsymbol{p}, \boldsymbol{x}) &= \sum_{j=1}^{d} \rho_j^3 + 3 \sum_{i \neq j} \rho_i \rho_j^2 + 6 \sum_{k>i>i} \rho_i \rho_j \rho_k \\ &= \mathcal{D}^3(\boldsymbol{p}, \boldsymbol{x}) + 3 \sum_{i \neq j} \rho_i \rho_j^2 + 6\mathcal{A}^3(\boldsymbol{p}, \boldsymbol{x}). \end{aligned}$$

We can compute the second term efficiently by using

$$\begin{aligned} \sum_{i \neq j} \rho_i \rho_j^2 &= \sum_{i,j=1}^{d} \rho_i \rho_j^2 - \sum_{j=1}^{d} \rho_j^3 \\ &= \mathcal{D}^{2,1}(\boldsymbol{p}, \boldsymbol{x}) - \mathcal{D}^3(\boldsymbol{p}, \boldsymbol{x}). \end{aligned}$$

We therefore obtain

$$\begin{aligned} \mathcal{A}^3(\boldsymbol{p}, \boldsymbol{x}) &= \frac{1}{6} \left[\mathcal{H}^3(\boldsymbol{p}, \boldsymbol{x}) - \mathcal{D}^3(\boldsymbol{p}, \boldsymbol{x}) - 3 \left(\mathcal{D}^{2,1}(\boldsymbol{p}, \boldsymbol{x}) - \mathcal{D}^3(\boldsymbol{p}, \boldsymbol{x}) \right) \right] \\ &= \frac{1}{6} \left[\mathcal{H}^3(\boldsymbol{p}, \boldsymbol{x}) - 3\mathcal{D}^{2,1}(\boldsymbol{p}, \boldsymbol{x}) + 2\mathcal{D}^3(\boldsymbol{p}, \boldsymbol{x}) \right]. \end{aligned}$$

B.3. Proof of multi-convexity (Theorem 1)

Let us denote the rows of $\boldsymbol{P}$ by $\bar{\boldsymbol{p}}_1, \ldots, \bar{\boldsymbol{p}}_d \in \mathbb{R}^k$. Using Lemma 2, we know that there exists constants a_s and b_s such that for all $j \in [d]$

$$\begin{aligned}
\hat{y}_{\mathcal{A}^m}(\boldsymbol{x}; \boldsymbol{\lambda}, \boldsymbol{P}) &= \sum_{s=1}^{k} \lambda_s \mathcal{A}^m(\boldsymbol{p}_s, \boldsymbol{x}) \\
&= \sum_{s=1}^{k} \lambda_s (p_{js} x_j a_s + b_s) \\
&= \sum_{s=1}^{k} p_{js} \lambda_s x_j a_s + \text{const} \\
&= \langle \bar{\boldsymbol{p}}_j, \bar{\boldsymbol{\mu}}_j \rangle + \text{const} \quad \text{where} \quad \bar{\boldsymbol{\mu}}_j \coloneqq [\lambda_1 x_j a_1, \ldots, \lambda_k x_j a_k]^{\mathrm{T}}.
\end{aligned}$$

Hence $\hat{y}_{\mathcal{A}^m}(\boldsymbol{x}; \boldsymbol{\lambda}, \boldsymbol{P})$ is an affine function of $\bar{\boldsymbol{p}}_1, \ldots, \bar{\boldsymbol{p}}_d$. The composition of a convex loss function and an affine function is convex. Therefore, (4) is convex in $\bar{\boldsymbol{p}}_j \; \forall j \in [d]$. Convexity w.r.t. $\boldsymbol{\lambda}$ is obvious.

C. Proof of equivalence between regularized problems (Theorem 2)

First, we are going to prove that the optimal solution of the nuclear norm penalized problem is a symmetric matrix. For that, we need the following lemma.

Lemma 7 *Upper-bound on nuclear norm of symmetrized matrix*

$$\|\mathcal{S}(\boldsymbol{M})\|_* \leq \|\boldsymbol{M}\|_* \quad \forall \boldsymbol{M} \in \mathbb{R}^{d^2}$$

Proof.

$$\begin{aligned}
\|\mathcal{S}(\boldsymbol{M})\|_* &= \|\frac{1}{2}(\boldsymbol{M} + \boldsymbol{M}^{\mathrm{T}})\|_* \\
&= \frac{1}{2}(\|\boldsymbol{M} + \boldsymbol{M}^{\mathrm{T}}\|_*) \\
&\leq \frac{1}{2}(\|\boldsymbol{M}\|_* + \|\boldsymbol{M}^{\mathrm{T}}\|_*) \\
&= \|\boldsymbol{M}\|_*,
\end{aligned}$$

with equality in the third line holding if and only if $\boldsymbol{M} = \boldsymbol{M}^{\mathrm{T}}$. The second and third lines use absolute homogeneity and subadditivity, two properties that matrix norms satisfy. The last line uses the fact that $\|\boldsymbol{M}\|_* = \|\boldsymbol{M}^{\mathrm{T}}\|_*$. □

Lemma 8 *Symmetry of optimal solution of nuclear norm penalized problem*

$$\operatorname*{argmin}_{\boldsymbol{M} \in \mathbb{R}^{d^2}} \bar{L}_{\mathcal{K}}(\boldsymbol{M}) \coloneqq L_{\mathcal{K}}(\mathcal{S}(\boldsymbol{M})) + \beta \|\boldsymbol{M}\|_* \in \mathbb{S}^{d^2}$$

Proof. From any (possibly asymmetric) square matrix $\boldsymbol{A} \in \mathbb{R}^{d^2}$, we can construct $\boldsymbol{M} = \mathcal{S}(\boldsymbol{A})$. We obviously have $L_{\mathcal{K}}(\mathcal{S}(\boldsymbol{A})) = L_{\mathcal{K}}(\mathcal{S}(\boldsymbol{M}))$. Combining this with Lemma 7, we have that $\bar{L}_{\mathcal{K}}(\boldsymbol{M}) \leq \bar{L}_{\mathcal{K}}(\boldsymbol{A})$. Therefore we can always achieve the smallest objective value by choosing a symmetric matrix. □

Next, we recall the variational formulation of the nuclear norm based on the SVD.

Lemma 9 *Variational formulation of nuclear norm based on SVD*

$$\|\boldsymbol{M}\|_* = \min_{\substack{\boldsymbol{U}, \boldsymbol{V} \\ \boldsymbol{M} = \boldsymbol{U}\boldsymbol{V}^{\mathrm{T}}}} \frac{1}{2}(\|\boldsymbol{U}\|_F^2 + \|\boldsymbol{V}\|_F^2) \quad \forall \boldsymbol{M} \in \mathbb{R}^{d^2} \tag{17}$$

Table 1. Examples of convex loss functions. We defined $\tau = \frac{1}{1+e^{-y\hat{y}}}$.

Loss	Domain of y	$\ell(y, \hat{y})$	$\ell'(y, \hat{y})$	$\ell''(y, \hat{y})$	μ
Squared	$\mathbb{R}$	$\frac{1}{2}(\hat{y} - y)^2$	$\hat{y} - y$	1	1
Squared hinge	$\{-1, 1\}$	$\max(1 - y\hat{y}, 0)^2$	$-2y \max(1 - y\hat{y}, 0)$	$2\delta_{[\hat{y}y<1]}$	2
Logistic	$\{-1, 1\}$	$\log(\tau^{-1})$	$y(\tau - 1)$	$\tau(1 - \tau)$	$\frac{1}{4}$

The minimum above is attained at $\|\boldsymbol{M}\|_* = \frac{1}{2}(\|\boldsymbol{U}\|_F^2 + \|\boldsymbol{V}\|_F^2)$*, where* $\boldsymbol{U} \in \mathbb{R}^{d \times r}$ *and* $\boldsymbol{V} \in \mathbb{R}^{d \times r}$*,* $r = \operatorname{rank}(\boldsymbol{M})$*, are formed from the reduced SVD of* $\boldsymbol{M}$*, i.e.,* $\boldsymbol{U} = \boldsymbol{A} \operatorname{diag}(\boldsymbol{\sigma})^{\frac{1}{2}}$ *and* $\boldsymbol{V} = \boldsymbol{B} \operatorname{diag}(\boldsymbol{\sigma})^{\frac{1}{2}}$ *where* $\boldsymbol{M} = \boldsymbol{A} \operatorname{diag}(\boldsymbol{\sigma}) \boldsymbol{B}^{\mathrm{T}}$*.*

For a proof, see for instance (Mazumder et al., 2010, Section A.5).

Now, we give a specialization of the above for symmetric matrices, based on the eigendecomposition instead of SVD.

Lemma 10 *Variational formulation of nuclear norm based on eigendecomposition*

$$\|\boldsymbol{M}\|_* = \min_{\substack{\boldsymbol{\lambda}, \boldsymbol{P} \\ \boldsymbol{M} = \boldsymbol{P} \operatorname{diag}(\boldsymbol{\lambda}) \boldsymbol{P}^{\mathrm{T}}}} \sum_{s=1}^{k} |\lambda_s| \, \|\boldsymbol{p}_s\|^2 \quad \forall \boldsymbol{M} \in \mathbb{S}^{d^2}, \tag{18}$$

where $k = \operatorname{rank}(\boldsymbol{M})$*. The minimum above is attained by the reduced eigendecomposition* $\boldsymbol{M} = \boldsymbol{P} \operatorname{diag}(\boldsymbol{\lambda}) \boldsymbol{P}^{\mathrm{T}}$ *and* $\|\boldsymbol{M}\|_* = \|\boldsymbol{\lambda}\|_1$*.*

Proof. Let $\boldsymbol{A} \operatorname{diag}(\boldsymbol{\sigma}) \boldsymbol{B}^{\mathrm{T}}$ and $\boldsymbol{P} \operatorname{diag}(\boldsymbol{\lambda}) \boldsymbol{P}^{\mathrm{T}}$ be the reduced SVD and eigendecomposition of $\boldsymbol{M} \in \mathbb{S}^{d^2}$, respectively. The relation between the SVD and the eigendecomposition is given by

$$\begin{aligned} \sigma_s &= |\lambda_s| \\ \boldsymbol{a}_s &= \operatorname{sign}(\lambda_s) \boldsymbol{p}_s \\ \boldsymbol{b}_s &= \boldsymbol{p}_s. \end{aligned}$$

From Lemma 9, we therefore obtain

$$\begin{aligned} \boldsymbol{u}_s &= \sqrt{\sigma_s} \boldsymbol{a}_s = \sqrt{|\lambda_s|} \operatorname{sign}(\lambda_s) \boldsymbol{p}_s \\ \boldsymbol{v}_s &= \sqrt{\sigma_s} \boldsymbol{b}_s = \sqrt{|\lambda_s|} \boldsymbol{p}_s. \end{aligned}$$

Now, computing $\frac{1}{2}(\sum_s \|\boldsymbol{u}_s\|^2 + \|\boldsymbol{v}_s\|^2)$ gives $\sum_{s=1}^{k} |\lambda_s| \, \|\boldsymbol{p}_s\|^2$. The minimum value $\|\boldsymbol{M}\|_* = \|\boldsymbol{\lambda}\|_1$ follows from the fact that $\boldsymbol{P}$ is orthonormal and hence $\|\boldsymbol{p}_s\|^2 = 1 \; \forall s \in [k]$. □

We now have all the tools to prove our result. The equivalence between (12) and (14) when $r = \operatorname{rank}(\boldsymbol{M}^*)$ is a special case of (Mazumder et al., 2010, Theorem 3). From Lemma 8, we know that the optimal solution of (14) is symmetric. This allows us to substitute (17) with (18), and therefore, (13) is equivalent to (14) with $k = \operatorname{rank}(\boldsymbol{M}^*)$. As discussed in (Mazumder et al., 2010), the result also holds when $r = k$ is larger than $\operatorname{rank}(\boldsymbol{M}^*)$.

D. Efficient coordinate descent algorithms

D.1. Direct approach, $\mathcal{K} = \mathcal{A}^m$ for $m \in \{2, 3\}$

As stated in Theorem 1, the direct optimization objective is multi-convex when $\mathcal{K} = \mathcal{A}^m$. This allows us to easily minimize the objective by solving a succession of coordinate-wise convex problems. In this section, we develop an efficient algorithm for minimizing (13) with $m \in \{2, 3\}$. It is easy to see that minimization w.r.t. $\boldsymbol{\lambda}$ can be reduced to a standard ℓ_1-regularized convex objective via a simple change of variable. We therefore focus our attention to minimization w.r.t. $\boldsymbol{P}$.

As a reminder, we want to minimize

$$f := \sum_{i=1}^{n} \ell(y_i, \hat{y}_i) + \beta \sum_{s=1}^{k} |\lambda_s| \|\boldsymbol{p}_s\|^2$$

where

$$\hat{y}_i \coloneqq \sum_{s=1}^{k} \lambda_s \mathcal{A}^m(\boldsymbol{p}_s, \boldsymbol{x}_i).$$

After routine calculation, we obtain

$$\begin{aligned}
\frac{\partial \mathcal{A}^2(\boldsymbol{p}_s, \boldsymbol{x}_i)}{\partial p_{js}} &= \langle \boldsymbol{p}_s, \boldsymbol{x}_i \rangle x_{ji} - p_{js} x_{ji}^2 = (\langle \boldsymbol{p}_s, \boldsymbol{x}_i \rangle - p_{js} x_{ji}) x_{ji} \\
\frac{\partial \mathcal{A}^3(\boldsymbol{p}_s, \boldsymbol{x}_i)}{\partial p_{js}} &= \frac{1}{2} \langle \boldsymbol{p}_s, \boldsymbol{x}_i \rangle^2 x_{ji} - p_{js} x_{ji}^2 \langle \boldsymbol{p}_s, \boldsymbol{x}_i \rangle - \frac{1}{2} x_{ji} \mathcal{D}^2(\boldsymbol{p}_s, \boldsymbol{x}_i) + p_{js}^2 x_{ji}^3 \\
&= \mathcal{A}^2(\boldsymbol{p}_s, \boldsymbol{x}_i) x_{ji} - p_{js} x_{ji}^2 \langle \boldsymbol{p}_s, \boldsymbol{x}_i \rangle + p_{js}^2 x_{ji}^3 \\
\frac{\partial \mathcal{A}^m(\boldsymbol{p}_s, \boldsymbol{x}_i)}{\partial p_{js}^2} &= 0 \quad \forall m \in \mathbb{N} \\
\frac{\partial \hat{y}_i}{\partial p_{js}} &= \lambda_s \frac{\partial \mathcal{A}^m(\boldsymbol{p}_s, \boldsymbol{x}_i)}{\partial p_{js}} \\
\frac{\partial \hat{y}_i}{\partial p_{js}^2} &= 0 \quad \forall j \in [d], s \in [k].
\end{aligned}$$

The fact that the second derivative is null is a consequence of the multi-linearity of $\mathcal{A}^m$.

Using the chain rule, we then obtain

$$\begin{aligned}
\frac{\partial f}{\partial p_{js}} &= \sum_{i=1}^{n} \ell'(y_i, \hat{y}_i) \frac{\partial \hat{y}_i}{\partial p_{js}} + 2\beta |\lambda_s| p_{js} \\
\frac{\partial f}{\partial p_{js}^2} &= \sum_{i=1}^{n} \left[\ell''(y_i, \hat{y}_i) \left(\frac{\partial \hat{y}_i}{\partial p_{js}} \right)^2 + \ell'(\hat{y}_i, y_i) \frac{\partial \hat{y}_i}{\partial p_{js}^2} \right] + 2\beta |\lambda_s| \\
&= \sum_{i=1}^{n} \ell''(y_i, \hat{y}_i) \left(\frac{\partial \hat{y}_i}{\partial p_{js}} \right)^2 + 2\beta |\lambda_s|.
\end{aligned}$$

Assuming that ℓ is μ-smooth, its second derivative is upper-bounded by μ and therefore we have

$$\frac{\partial f}{\partial p_{js}^2} \leq \eta_{js} \quad \text{where} \quad \eta_{js} \coloneqq \mu \sum_{i=1}^{n} \left(\frac{\partial \hat{y}_i}{\partial p_{js}} \right)^2 + 2\beta |\lambda_s|.$$

Then the update

$$p_{js} \leftarrow p_{js} - \eta_{js}^{-1} \frac{\partial f}{\partial p_{js}}$$

guarantees that the objective value is monotonically decreasing except at the coordinate-wise minimum. Note that in the case of the squared loss $\ell(y, \hat{y}) = \frac{1}{2}(y - \hat{y})^2$, the above update is equivalent to a Newton step and is the exact minimizer of the coordinate-wise objective. An epoch consists in updating all variables once, for instance in cyclic order.

For an efficient implementation, we need to maintain $\hat{y}_i \ \forall i \in [n]$ and statistics that depend on $\boldsymbol{p}_s$. For the former, we need $O(n)$ memory. For the latter, we need $O(kmn)$ memory for an implementation with full cache. However, this requirement is not realistic for a large training set. In practice, the memory requirement can be reduced to $O(mn)$ if we recompute the quantities then sweep through $p_{1s}, \ldots, p_{ds}$ for s fixed. Overall the cost of one epoch is $O(kn_z(\boldsymbol{X}))$. A similar implementation technique is described for factorization machines with $m = 2$ in (Rendle, 2012).

D.2. Lifted approach, $\mathcal{K} = \mathcal{H}^m$

We present an efficient coordinate descent solver for the lifted approach with $\mathcal{K} = \mathcal{H}^m$, for arbitrary integer $m \geq 2$. Recall that our goal is to learn $\boldsymbol{\mathcal{W}} = \mathcal{S}(\boldsymbol{\mathcal{M}}) \in \mathbb{S}^{d^m}$ by factorizing $\boldsymbol{\mathcal{M}} \in \mathbb{R}^{d^m}$ using m matrices of size $d \times r$. Let us call these

Algorithm 1 CD algorithm for direct obj. with $\mathcal{K} = \mathcal{A}^{\{2,3\}}$

Input: $\boldsymbol{\lambda}$, initial $\boldsymbol{P}$, μ-smooth loss function ℓ, regularization parameter β, number of bases k, degree m, tolerance ϵ
Pre-compute $\hat{y}_i \coloneqq \hat{y}_{\mathcal{A}^m}(\boldsymbol{x}_i; \boldsymbol{\lambda}, \boldsymbol{P})\ \forall i \in [n]$
Set $\Delta \leftarrow 0$
for $s \coloneqq 1, \ldots, k$ **do**
  Pre-compute $\langle \boldsymbol{p}_s, \boldsymbol{x}_i \rangle$ and $\mathcal{A}^2(\boldsymbol{p}_s, \boldsymbol{x}_i)\ \forall i \in [n]$
  for $j \coloneqq 1, \ldots, d$ **do**
    Compute inv. step size $\eta \coloneqq \mu \sum_{i=1}^n \left(\frac{\partial \hat{y}_i}{\partial p_{js}}\right)^2 + 2\beta|\lambda_s|$
    Compute $\delta \coloneqq \eta^{-1}\left[\sum_{i=1}^n \ell'(y_i, \hat{y}_i)\frac{\partial \hat{y}_i}{\partial p_{js}} + 2\beta|\lambda_s| p_{js}\right]$
    Update $p_{js} \leftarrow p_{js} - \delta$; Set $\Delta \leftarrow \Delta + |\delta|$
    Synchronize $\hat{y}_i$, $\langle \boldsymbol{p}_s, \boldsymbol{x}_i \rangle$ and $\mathcal{A}^2(\boldsymbol{p}_s, \boldsymbol{x}_i)\ \forall i$ s.t. $x_{ji} \neq 0$
  end for
end for
If $\Delta \leq \epsilon$ stop, otherwise repeat
Output: $\boldsymbol{P}$

Algorithm 2 CD algorithm for lifted objective with $\mathcal{K} = \mathcal{H}^m$

Input: initial $\{\boldsymbol{U}^t\}_{t=1}^m$, μ-smooth loss function ℓ, regularization parameter β, rank r, degree m, tolerance ϵ
Pre-compute $\hat{y}_i \coloneqq \sum_{s=1}^r \prod_{t=1}^m \langle \boldsymbol{u}_s^t, \boldsymbol{x}_i \rangle\ \forall i \in [n]$
Set $\Delta \leftarrow 0$
for $t \coloneqq 1, \ldots, m$ and $s \coloneqq 1, \ldots, r$ **do**
  Pre-compute $\xi_i \coloneqq \prod_{t' \neq t} \langle \boldsymbol{u}_s^{t'}, \boldsymbol{x}_i \rangle\ \forall i \in [n]$
  for $j \coloneqq 1, \ldots, d$ **do**
    Compute inv. step size $\eta \coloneqq \mu \sum_{i=1}^n \xi_i^2 x_{ji}^2 + \beta$
    Compute $\delta \coloneqq \eta^{-1}\left[\sum_{i=1}^n \ell'(y_i, \hat{y}_i)\xi_i x_{ji} + \beta u_{js}^t\right]$
    Update $u_{js}^t \leftarrow u_{js}^t - \delta$; Set $\Delta \leftarrow \Delta + |\delta|$
    Synchronize $\hat{y}_i\ \forall i$ s.t. $x_{ji} \neq 0$
  end for
end for
If $\Delta \leq \epsilon$ stop, otherwise repeat
Output: $\{\boldsymbol{U}^t\}_{t=1}^m$

matrices $\boldsymbol{U}^1, \ldots, \boldsymbol{U}^m$ and their columns $\boldsymbol{u}_s^t = [u_{1s}^t, \ldots, u_{ds}^t]^{\mathrm{T}}$ with $t \in [m]$ and $s \in [r]$. The decomposition of $\boldsymbol{\mathcal{M}}$ can be expressed as a sum of rank-one tensors

$$\boldsymbol{\mathcal{M}} = \sum_{s=1}^r \boldsymbol{u}_s^1 \otimes \cdots \otimes \boldsymbol{u}_s^m.$$

Using (11) we obtain

$$\hat{y}_i \coloneqq \langle \boldsymbol{\mathcal{W}}, \boldsymbol{x}_i^{\otimes m} \rangle = \langle \boldsymbol{\mathcal{M}}, \boldsymbol{x}_i^{\otimes m} \rangle = \sum_{s=1}^r \prod_{t=1}^m \langle \boldsymbol{u}_s^t, \boldsymbol{x}_i \rangle.$$

The first and second coordinate-wise derivatives are given by

$$\frac{\partial \hat{y}_i}{\partial u_{js}^t} = \prod_{t' \neq t} \langle \boldsymbol{u}_s^{t'}, \boldsymbol{x}_i \rangle x_{ji} \quad \text{and} \quad \frac{\partial \hat{y}_i}{\partial (u_{js}^t)^2} = 0.$$

We consider the following regularized objective function

$$f \coloneqq \sum_{i=1}^n \ell(y_i, \hat{y}_i) + \frac{\beta}{2} \sum_{t=1}^m \sum_{s=1}^r \|\boldsymbol{u}_s^t\|^2.$$

Using the chain rule, we obtain

$$\frac{\partial f}{\partial u_{js}^t} = \sum_{i=1}^n \ell'(y_i, \hat{y}_i)\frac{\partial \hat{y}_i}{\partial u_{js}^t} + \beta u_{js}^t \quad \text{and} \quad \frac{\partial f}{\partial (u_{js}^t)^2} = \sum_{i=1}^n \ell''(y_i, \hat{y}_i)\left(\frac{\partial \hat{y}_i}{\partial u_{js}^t}\right)^2 + \beta.$$

Assuming that ℓ is μ-smooth, its second derivative is upper-bounded by μ and therefore we have

$$\frac{\partial f}{\partial (u_{js}^t)^2} \leq \eta_{js}^t \quad \text{where} \quad \eta_{js}^t \coloneqq \mu \sum_{i=1}^n \left(\frac{\partial \hat{y}_i}{\partial u_{js}^t}\right)^2 + \beta.$$

Then the update

$$u_{js}^t \leftarrow u_{js}^t - (\eta_{js}^t)^{-1} \frac{\partial f}{\partial u_{js}^t}$$

guarantees that the objective value is monotonically decreasing, except at the coordinate-wise minimum. Note that in the case of the squared loss $\ell(y, \hat{y}) = \frac{1}{2}(y - \hat{y})^2$, the above update is equivalent to a Newton step and is the exact minimizer of the coordinate-wise objective. An epoch consists in updating all variables once, for instance in cyclic order.

For an efficient implementation, the two quantities we need to maintain are $\hat{y}_i \; \forall i \in [n]$ and $\prod_{t' \neq t} \langle \boldsymbol{u}_s^{t'}, \boldsymbol{x}_i \rangle \; \forall i \in [n], s \in [r], t \in [m]$. For the former, we need $O(n)$ memory. For the latter, we need $O(rmn)$ memory for an implementation with full cache. However, this requirement is not realistic for a large training set. In practice, the memory requirement can be reduced to $O(mn)$ if we recompute the quantity then sweep through $u_{1s}^t, \ldots, u_{ds}^t$ for t and s fixed. Overall the cost of one epoch is $O(mrn_z(\boldsymbol{X}))$.

D.3. Lifted approach, $\mathcal{K} = \mathcal{A}^2$

For $\langle \cdot, \cdot \rangle_>$, efficient computations are more involved since we need to ignore irrelevant monomials. Nevertheless, we can also compute the predictions directly without explicitly symmetrizing the model. For $m = 2$, it suffices to subtract the effect of squared features. It is easy to verify that we then obtain

$$\langle \mathcal{S}(\boldsymbol{U}\boldsymbol{V}^{\mathrm{T}}), \boldsymbol{x}^{\otimes 2} \rangle_> = \frac{1}{2} \left[\langle \boldsymbol{U}^{\mathrm{T}}\boldsymbol{x}, \boldsymbol{V}^{\mathrm{T}}\boldsymbol{x} \rangle - \sum_{s=1}^{r} \langle \boldsymbol{u}_s \circ \boldsymbol{x}, \boldsymbol{v}_s \circ \boldsymbol{x} \rangle \right],$$

where $\circ$ indicates element-wise product. The coordinate-wise derivatives are given by

$$\frac{\partial y_i}{\partial u_{js}} = \frac{1}{2} \left[\langle \boldsymbol{v}_s, \boldsymbol{x} \rangle x_{ji} - v_{js} x_{ji}^2 \right] \quad \text{and} \quad \frac{\partial y_i}{\partial v_{js}} = \frac{1}{2} \left[\langle \boldsymbol{u}_s, \boldsymbol{x} \rangle x_{ji} - u_{js} x_{ji}^2 \right].$$

Generalizing this to arbitrary m is a future work.

E. Datasets

For regression experiments, we used the following public datasets.

Dataset	n (train)	n (test)	d	Description
abalone	3,132	1,045	8	Predict the age of abalones from physical measurements
cadata	15,480	5,160	8	Predict housing prices from economic covariates
cpusmall	6,144	2,048	12	Predict a computer system activity from system performance measures
diabetes	331	111	10	Predict disease progression from baseline measurements
E2006-tfidf	16,087	3,308	150,360	Predict volatility of stock returns from company financial reports

The *diabetes* dataset is available in scikit-learn (Pedregosa et al., 2011). Other datasets are available from `http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/`.

For recommender system experiments, we used the following two public datasets.

Dataset	n	d
Movielens 1M	1,000,209 (ratings)	9,940 = 6,040 (users) + 3,900 (movies)
Last.fm	108,437 (tag counts)	24,078 = 12,133 (artists) + 11,945 (tags)

For *Movielens 1M*, the task is to predict ratings between 1 and 5 given by users to movies, i.e., $y \in \{1, \ldots, 5\}$. For *Last.fm*, the task is to predict the number of times a tag was assigned to an artist, i.e., $y \in \mathbb{N}$.

The design matrix $\boldsymbol{X}$ was constructed following (Rendle, 2010; 2012). Namely, for each rating y_i, the corresponding $\boldsymbol{x}_i$ is set to the concatenation of the one-hot encodings of the user and item indices. Hence the number of samples n is the number of ratings and the number of features is equal to the sum of the number of users and items. Each sample contains exactly two non-zero features. It is known that factorization machines are equivalent to matrix factorization when using this representation (Rendle, 2010; 2012).

We split samples uniformly at random between 75% for training and 25% for testing.